

Accelerating Module Conditioning

Local application for new Linac

Sep 24, 1992

Introduction

Conditioning the new Linac rf cavities will take hours or days for each module. This local application is an implementation designed to automate the procedure so that it can run unattended. The main idea is to slowly ramp the rf peak forward power from the klystron toward a target value, as long as the radiation remains below a threshold, taking care to watch the vacuum pressure and the spark rate, reducing the forward power if either exceeds specified levels, and resetting any rf system trips. The implementation of the program is the subject of this note. The user interface is provided by the Parameter Page.

Hardware signals (klystron system 3)

| <i>Description</i> | <i>Name</i> | <i>Units</i> |
|--|---------------|--------------|
| 1. rf peak forward power A/D reading | K3WG1P | MW |
| 2. rf peak forward power D/A setting | " | MW |
| 3. vacuum pressure A/D reading (2) | V3VP1 , V3VP2 | V |
| 4. spark digital status bit reading | | |
| 5. rf "on" digital status bit reading | | |
| 6. rf interlocks reset digital control bit pulse | | |
| 7. rf system reset digital control bit pulse | | |
| 8. clock event status bits | | |

Software parameters

| <i>Description</i> | <i>Name</i> | <i>Units</i> |
|---|-------------|--------------|
| 1. enable status/control bit for this application | | |
| 2. rf peak forward power target value | K3TRGP | MW |
| 3. rf peak forward power delta | K3DLTP | MW |
| 4. time interval delta | K3DLTT | SEC |
| 5. spark rate threshold | K3SPKT | % |
| 6. vacuum pressure threshold | K3VACT | V |
| 7. rf peak power back-off percent | K3PPBK | % |
| 8. maximum #resets of trips | K3MAXT | |
| 9. spark rate output value | K3SPKR | % |
| 10. delay after back-off due to vacuum | K3VACD | SEC |
| 11. maximum #sparks to compute spark rate | K3MAXS | |
| 12. status bit for program state 0. Waiting for recovery. | | |
| 13. control bit to clear spark-counting statistics counters | | |

Local application support

As a local application, the code is called as a Pascal procedure by a special entry in the Data Access Table of the local station. This entry causes each local application residing in the system Local Application Table be called by name. The 4-character name is used to search the CODES table of programs that have been previously downloaded by name into non-volatile memory. The first argument of the call is a byte whose value identifies the type of call: 0:

Initialization call. Allocate and initialize static memory used during the time the local application is enabled.

1: Termination call. Free static memory allocated by Initialization call.

2: (not used)

3: Cycle call. Process with new data in data pool.

4: Network call. Message received for this application.

The second argument is a pointer to the 12-word parameter area of the Local Application Table entry. The first *longword* of this area provides storage for the pointer to the static memory allocated during the Initialization call. The next *word* is the enable Bit#, and the remaining words are used for additional parameters, usually specified as Channel#s and Bit#s. (See the layout for this application in a later section.) When the enable bit is set, the application is enabled. When it is clear, the application is disabled. The system notices changes in the state of this enable bit and schedules Initialization and Termination calls accordingly. When there is no change in the enable bit, *and the bit is set*, the application receives a Cycle call. Special logic is included that provides for automatic replacement to a new program version as soon as it is downloaded, if the application is enabled. A local application is downloaded into non-volatile memory but executes out of on-board ram. A checksum is kept for the downloaded version that is verified each time an application's enable bit changes from a 0 to a 1 and its code is copied into allocated ram for execution.

State flow

Local applications of the closed loop style are typically implemented with state logic. In this case of the cavity conditioning application, there are two states: 0 and 1. When the application is first enabled, state 0 is asserted. While in state 0, the application looks for a valid set of readings (both hardware and software) and also for the rf system to be "on". Constants in the program are used to assess whether the values of the hardware and software parameters are within "reasonable" ranges. Once these conditions are satisfied, the program switches to state 1.

In state 1, the time delta value is used as a period over which to determine a maximum value of the rf peak forward power readings. At the end of the time interval, the maximum is compared with the target value to determine whether an adjustment of the peak power delta can bring the power closer to the target

value. Independent of this time delta interval, the maximum #sparks parameter is used to form a spark period interval over which the spark rate is calculated. The spark rate is checked against the threshold value to decide whether to back off. Vacuum is always checked against the vacuum threshold to decide whether to back off. And the rf "on" status is always checked to detect rf system trips.

Statistics are maintained about the relationship between sparks that occur and the relevant Tevatron clock event signals that indicate what kind of accelerating cycle was active. Counts of events, sparks occurring on those events, counts of sparks that occur during which Booster batches, and a histogram of spark occurrences throughout the supercycle. A self-clearing control bit is provided that, when set, clears the spark counting statistics.

All software input parameters (except the maximum #trips) can be modified during normal state 1 operation to take effect after the current time interval. Also, the peak forward power can be changed manually by knob control even while the conditioning program is regulating the peak power, as the changes made by the algorithm are always applied incrementally from the current setting.

A "state 0" status bit is provided as an output so that it can be monitored by the alarm system to announce when the application is no longer regulating.

Logic details

The response to an rf system trip is to first back off the forward power, reset the rf interlocks and subsequently to reset the rf system itself. Such resets of rf trips are limited to repeat no more often than 10 seconds, a program constant.

There are two vacuum readings. The one with the worst reading is used in the algorithm because only one vacuum pump may be required to be running, and the reading of a pump which is off appears as "excellent" vacuum. The response to poor vacuum compared to the vacuum threshold value is to back off the forward power and delay for the vacuum delay time before allowing another back-off due to vacuum. This gives the vacuum system time to approach equilibrium under operation at a reduced peak forward power level.

The spark rate is computed over the number of 15 Hz cycles required to accumulate the number of sparks specified by the maximum #sparks parameter. Expressed as a percentage, it is compared against the spark threshold.

When there have been more rf system trips than that given by the maximum #trips parameter, the application reverts to state 0. Manual recovery of the rf system will allow a return to state 1 processing with an additional maximum

#trips permitted—assuming nothing else is wrong.

The application was developed using MPW Pascal on the Macintosh to take advantage of its support for the floating point 68881 chip, as most of the logic in the program is based upon engineering units values. Its current 400 lines of source code run in less than 3K bytes.

Parameters layout

The layout of the parameters area of the Local Application entry as viewed by the Local Application Parameter Page is as follows:

```
E LOC APPL PARAMS 09/24/92 1711
NODE<0623>  NTRY< 5>
NAME=COND  CNTR=00F8
TITL"LGAL RF CAV CONDITIONING"
SVAR=00000000
ENABLE  B<02A0> COND ENABLE
SPARKS  B<019E> REFLECTED POWER
RFONST  B<0198> RF ON IS ENABLED
RFINTLK B<015C> INTERLOCK RESET
PPWR    C<0480> K3WG1P      MW
VACUUM  C<049E> V3VP1      V
          <0000>
RFRESET B<0325> SYSTEM RESET
EVENTS  B<0230> EVENT 18
OTHERS  C<0490> K3TRGP      MW
```

The first parameter word specifies the enable Bit# that must be set to enable the local application program to run. Setting it to a “1” enables calls to be made to the application, beginning with the Initialization call. Setting it to a “0” schedules a Termination call before releasing the program’s execution memory.

This application uses enable Bit#+1 for the “state 0” output bit and enable Bit#+2 for the control bit that clears the spark counters.

The lo byte of the CNTR word is merely a diagnostic count of the number of times the application code is called. It serves to show evidence of obvious activity when viewing this entry on a memory page display. The hi byte of the same word shows the elapsed time required by the last 15 Hz program invocation.

The ptr to the application’s static memory, established as a result of the standard Pascal New procedure used for dynamic memory allocation, is stored during Initialization call processing for use during subsequent Cycle call processing. One can gain some diagnostic insight of the application’s activity by observing the contents of the memory block pointed to by this address with the Memory Dump Page application. The first 8 bytes are used for a standard memory block header. The rest of the block can be matched with the declaration of the static variable data structure of the application.

The last 9 words are used for hardware Chan and Bit#. The final word is a base Chan# of the sequence of Chan#s used for the software application parameters.

ACNAUX Functions

Acnet utility support

Mar 6, 1992

Introduction

The Acnet standard for task-task communications is widely used in accelerator control systems at Fermilab. The Acnet task called ACNAUX is designed to support a set of utility functions which can aid diagnosis of Acnet network nodes in a standard way, as it is independent of the cpu and operating system used by each node. Each node supports the Acnet header standard; through ACNAUX each node supports some common diagnostic utility functions. This note describes the support for ACNAUX in the Local Station nodes. The official document of the standard is described elsewhere by Glenn Johnson.

Each function is specified by the lo byte of the first (or only) word following the Acnet header of the request message. In some cases the hi byte of this word may be used as a sub-function code. All functions are one-shot requests.

ACNAUX implementation

In the local station, ACNAUX is implemented by a local application called AAUX. It uses the generic protocol support available via OpenPro to receive notification about network messages directed to it. This same support is also used by FTPMAN, GATE, and HUMBUG. It permits more rapid response than would be achieved using 15 Hz polling of the message queue. Two of the available parameter words are used to pass a ptr to the message reference block that itself includes a ptr to the received request message. The AcReq Task calls the local application when it receives a message destined for the local application whose network task name is found in the protocol table, filled by OpenPro calls.

NOOP function 0

This serves as a "ping" facility. It determines if a node will respond to an Acnet header request message. A status-only reply is returned. A Vax program called ANPING can be run from a terminal to exercise this function. It includes the time for the response in 10 msec resolution. A local station which is not busy can return such a response in 4 msec, which is near the limit of the token ring chipset that interfaces to the token ring network.

GTTASK function 4

Returns a list of the currently-connected network task names, followed by a byte array of the associated task-id's. The AAUX local application examines the NETCT table contents to find this info. For each entry whose queue id is nonzero, the task name and id is recorded. Because the format is specified in Vax normal byte order, it is necessary to swap bytes for all words in the reply.

The byte order of task names used in the local stations was designed to conform to the notion that a task name can be a 4-byte character array. But in the acnet system, many task names are in 6-character RAD-50 format, which also takes 4 bytes (two 3-character words). (Recall for the following argument that the token ring hardware interface on the Vax swaps every byte, in order to make it such that 2-byte integer words transfer between Vax and token ring stations that use a "big-endian" architecture without software byte swapping.) To make it possible for both the Vax and the local station to use 4-character ascii names, the bytes of the destination task name field of an acnet header are swapped upon reception by the ANet Task in the local station. ANet then searches for a match with the current connected task entries in the NETCT net connection table to dispatch the received message to the proper message queue. When a message is transmitted by the local station, these bytes are swapped before it gets passed to the chipset so that the Vax receives them in natural order.

As a result of this logic of preserving 4-char ascii task name communication, the 6-character RAD-50 names must be kept in byte-swapped form in the NETCT table. Since these names are treated as magic constants by local station software, this is easy to do. As an example, the task name ACNAUX in RAD-50 form is \$06C609A0 (ACN=06C6, AUX=09A0); but for local station software, it should be specified as \$C606A009, and it appears this way in the NETCT table entry.

Since there can be a mix of 4-character and 6-character formats, it requires some special logic to convert the names to ascii for display. All 4-character task names are composed of 4 capital letters in ascii. If a given task name fits this pattern, then it may be presumed a 4-character form; otherwise, it should be assumed to be of the 6-character RAD-50 form.

RAD-50 definition

This encoding of a restricted set of characters permits squeezing 3 characters of information into one 16-bit word. It can be considered simply as a base-40 number system, whose coding scheme is as follows:

| | | | |
|------|-------|-------|----------|
| 0 | space | 28 | . |
| 1-26 | A-Z | 29 | (unused) |
| 27 | \$ | 30-39 | 0-9 |

To convert 'XYZ' into RAD-50, the result is $(25*40 + 26)*40 + 27 = 41067 = \$A06B$.

GTTRIO function 8

Returns token ring chipset I/O error statistics. The token ring chipset maintains an error log that is a set of nine 8-bit counters. A special command can be issued to the chipset to interrogate these counters. An extra motivation for doing so is provided by the fact that for some error conditions, when the error count reaches 255, or \$FF, the chipset removes itself from the network. This means that a node on token ring should plan to read this error log on some periodic basis. The local station software does this, using a default period of about 20 minutes,

currently. The counts are accumulated into a corresponding set of 16-bit counts, which allow monitoring the health of the network. This GTTRIO function returns the value of these word counts, along with the time interval over which they were accumulated. The names of the error conditions are:

Line

Each frame that is received or repeated for a valid FCS or Manchester code violation. If one is detected, the EDI (Error Detected Indicator) bit is set to "1" in the frame or token's ending delimiter. If the received EDI is "1", this Line error count is incremented; if the EDI is a "1", it is not incremented.

ARI/FCI

This indicates that the up-stream node chipset is unable to set its ARI/FCI bits in a frame it has received. (The details of this seem rather obscure to this writer.)

Burst

The chipset has detected the absence of transitions for five half-bit times between SDEL and EDEL.

Receive congestion

The chipset recognizes a frame addressed to its specific address, but it has no buffer space available to receive the frame.

Lost frame

When in transmit mode, the chipset fails to receive the end of the frame it has transmitted.

Frame copied

When in receive/repeat mode, the chipset recognizes a frame that is addressed to its specific address, but the ARI bits are nonzero, indicating a possible duplicate address. (The bridge currently causes many of these.)

Token

The Active Monitor detects a frame with the MONITOR COUNT bit set, no token of frame received within a 10 msec window, or a code violation in a starting delimiter/token sequence.

DMA parity or DMA Bus

Maybe something wrong with the token ring interface board itself.

GTPKTS function 9

Returns network message packet processing statistics to permit assessment of a node's network I/O activity. The time since the network statistics were cleared is given along with a count of message packets processed either in or out. For the local station, several resident diagnostic counters are monitored to collect these statistics. The time period is the time since the AAUX local application was last initialized, which would normally be at system reset time; however, if AAUX is updated to a new version, upon download of a new version, the old version is terminated and the new version is initialized, so the statistics will begin again. The implementation uses the cycle counter which is a longword that begins at

zero at system reset time and is incremented for every 15 Hz cycle. If a station is running at the backup 12.5 Hz rate, this value is not corrected for it. When AAUX is initialized, it captures the present reading of this cycle counter. To reply to a GTPKTS function request, the current cycle counter – the earlier one is returned.

The count of packets processed is fairly involved. The local station supports network communications with several protocols. The Classic data request/alarm pro to col does not use an acnet header. The DZero and Accelerator protocols do use an acnet header. Each family of protocols must be considered separately.

The Classic protocol uses SAP 18. At present, there is no message counter accumulated for the Classic protocol messages received, so a frame counter is monitored as an approximation. In the SAP table, a word counter is incremented for each SAP 18 frame received. AAUX watches this counter every cycle and notices changes in it to build a count of Classic frames received. When a message count is added to the system logic, this code can be updated to use it.

All Acnet-header protocols use SAP 68. Each task name that is connected to the network is recorded in the NETCT table, and a word counter is incremented for each message that is received and dispatched by the ANet Task. So AAUX monitors these counters for all 23 possible entries in NETCT. For each entry that is active (queue id $\neq 0$) the associated word counter is monitored every cycle for evidence of counting. Increments are accumulated into the total packet count.

All messages transmitted pass through the OUTPQ network output pointer queue. There is a word in the OUTPQ header that is incremented for every message that has been completely transmitted. This word is monitored every cycle and any increments noticed are accumulated into the total packet count.

All in all, the total packet count is the sum of the number of Classic frames received (to be replaced by a message count when available), the number of messages passed to the associated message queue for each connected network task name, and the number of messages completely transmitted to the network for any protocol.

Each Linac local station uses Arcnet communications for data acquisition with the SRMs (Smart Rack Monitors), which usually number 4 or 5 on each Arcnet. This communication protocol is Acnet-header based and is called locally “#4” to signify it was the 4th data request protocol to be supported by the local station system software. This network activity is not part of the token ring network activity; therefore, even though it represents network processing activity in the local station, it was *not* included in the #packets reported in reply to GTPKTS. If it were, it would typically add 75 packets per second for a station with 4 SRMs. This includes 1 broadcast transmitted request and 4 replies per 15 Hz cycle.

Viewing the results of these functions

One program that makes use of the GTPKTS function are Vax console page D31, which polls a large sequence of nodes and reports the total time value and the number of network packets processed per second between polls. Any node that does not support the GTPKTS function is sent a NOOP function instead, in which case only an indication of the success of the reply is reported.

Another program that exercises this function is called PACKETS, accessible from a VT100 terminal or emulator. It shows the network activity relating to 4 nodes, with the last one initially set to ADCALC. This last one can be changed to a user selected node name by typing "!" to get the prompt message that asks for the node name, such as LIN611, for example. When a user-selected name is entered, the GTTASK function is issued to request the task list, displayed in a separate box. Each node is polled approximately every 4 seconds, and statistics are displayed for the current packet rate, the minimum and maximum rates, and rates that are averaged over several different time intervals.

The program that uses the GTTRIO function to list the error counters at a VT100 terminal is called TRIO. It prompts for a node name and shows the current counts obtained via the GTTRIO function and also the time over which the counts were accumulated. At this time, the CLEAR and NOW and set new period are not implemented, but they could be so in the future if needed.

Alarm-based Signals

Local application

Wed, Jan 10, 1996

This **ASIG** local application is designed to support digital control line signals that reflect the alarm state of analog channels. If a channel is bad, one can assert some digital control signal that is otherwise de-asserted. One particular case like this that has always been supported is the beam inhibit control signal. Any analog channel, or binary status bit that is in the alarm scan may have the optional attribute of causing the beam inhibit signal to be asserted when it is bad. For example, if a Linac RF gradient is out of tolerance, it is desirable to automatically prevent beam from being accelerated through the Linac, as it is not good beam anyway. This local application supports the same service, but applied to other control lines that may be needed to affect equipment in specific ways.

The parameters of the **ASIG** local application appear as follows:

```
E LOCAL APPS      01/10/96 1120
NODE<0614>  NTRY<32>/64  H<0508>
NAME=ASIG  CNTR=AD  DT= 0    MS
TITL"ALARM-BASED CTRL SIGNAL "
SVAR=00000000      01/03/96 1508
ENABLE  B<00AA> ASIG ENABLE
SPARE    <0000>
CHAN1    C<011C> RF3HV   38.56 KV
C-BIT1   B<0191>*RF3  COMPUTR ENBL
CHAN2    C<031C> RF4HV   38.85 KV
C-BIT2   B<0391>*RF4  COMPUTR ENBL
CHAN3    C<051C> RF5HV   40.27 KV
C-BIT3   B<0591>*RF5  COMPUTR ENBL
CHAN4    C<0000>
C-BIT4   B<0000>
```

Up to four Channel-Bit pairs may be specified. If the Bit# of a pair is zero, that pair is inactive. Each pair logically acts independently. If the reading of the specified analog Channel is in alarm (in the "bad" state) then the specified control Bit# is asserted. The actual state of "asserted" is specified by the sign bit of the Bit# parameter. In the above example, the asserted state of each of the three specified controls lines is zero. As long as the RF3 High Voltage reading is in alarm, for example, the RF3 Computer Enable control signal is set to a zero; otherwise, it is set to a one. (If the reverse control signal logic had been desired, then the Bit# parameter would have been 8191 rather than 0191.)

Beam Summing

Local application

Fri, Jun 17, 1994

For monitoring long term accumulations of raw data such as beam charge, it is necessary to do the pulse-by-pulse summing locally, then making the accumulations available to any host system. This note describes an implementation for this as a local application.

Parameters layout:

```
ENABLE  B<00C4> BSUM ENABLE
BEAM    B<009F> NO BEAM STATUS
OUTADDR <2F00>
CHAN1   C<001B>   BEAM ACCUM TEST
CHAN2   C<0000>
CHAN3   C<0000>
CHAN4   C<0000>
```

The ENABLE Bit# enables operation of the BSUM local application. The BEAM status Bit# indicates what bit signals the presence of a scheduled beam pulse of the type summed by this application. The “beam” state is the sign bit of this parameter. The OUTADDR parameter is a 16-bit address in low memory where the output results are written. This address should be in an area that is zeroed at system reset time, in order to start the accumulations at zero and to signal a reset has occurred. (Consult with an expert to determine an appropriate address to use.) The rest of the parameters are analog channel#s whose readings provide the raw data to be summed. A zero channel# is ignored, but it occupies a “slot” in the output data structure.

The data structure of the accumulated data is as follows:

```
sum: ARRAY[1..4] OF Integer; { 4-word sum of beam data }
cnt: ARRAY[1..2] OF Integer; { 2-word sum of beam cycles }
tot: ARRAY[1..2] OF Integer; { 2-word sum of all cycles }
```

(Here, an “Integer” is a 16-bit word.) For each channel specified, a “slot” of 8 words is used starting at OUTADDR. The SSDN of the Acnet database entry for the reading property can include this OUTADDR. For example, if the source node were node 61E, and the base address of the data structure were 00002F00, one could then have the following SSDN structure: 1D02/061E/0000/2F00.

Upon reading this data, a host program can compute the accumulation (as a double precision floating point value) of each signal as follows, where $k=32768$:

```
acc:= ((sum[1]*k + sum[2])*k + sum[3])*k + sum[4];  
nBP:= cnt[1]*k + cnt[2];  
all:= tot[1]*k + tot[2];
```

These values need to be referenced to the set of values obtained during the last query by the host program, in order to get the amount of beam that has been accumulated over the last query interval. Upon conversion to engineering units, the data can be archived as needed. Note that this scheme works for multiple users without conflict.

The front end keeps 15 bits of precision in each word in order to maintain positive values that simplify the above formulas in hi-level language. (It also avoids the word-swap problems that can result from differences with Vax data formats, since words are automatically byte-swapped by Fermilab networking hardware.) Note that a 60-bit long summation will never overflow in anyone's lifetime. Also, 30 bits of pulses at 15 Hz is more than 2 years. Site-wide power outages occur more often than that. A reset of the front end clears the accumulation area.

The host program will zero its own version of the accumulations according to its particular implementation. If the host program finds that the accumulation has dropped to a lower value than it had during its previous query, it can assume that the system has reset and restarted its accumulations. The most beam accumulation that could have been lost would be that which occurred since the last query. Assuming a one-minute query interval, for example, this should not be significant. (Typical times between resets of Linac front end stations are measured in months.)

DAC0 and DAC1 Closed Loops

MR RF Local Applications

Mon, Apr 19, 1993

The two local applications DAC0 and DAC1 perform simple closed loop algorithms needed in the Main Ring RF systems. DAC0 regulates the injection offset for the ferrite bias supply. DAC1 regulates the high energy offset for the ferrite bias supply.

DAC0

At a certain time in the rf cycle (of length 2.5 seconds, in the case of collider operation), about 50 μ s after the phase detector trigger time, the phase detector error signal is sampled. If the value is more than 0.5 volts away from the nominal 0.0 volt level, then adjust the controlling D/A by 80 mv per volt of error to correct for the error. The adjustment is in an algebraically positive direction for a positive error.

A bit is set on the occurrence of the phase detector trigger signal, and the sample and hold circuit is triggered 50 μ s after that time to measure the error signal.

DAC1

When the rf turns off each cycle, measure the change in the ferrite bias supply current waveform. If the bias supply program is more than 2.5 volts, and if the bias supply is on and the modulator is on, then adjust the high energy bias supply offset. The amount of the offset is proportional to the measured change and is only to be made if the change is greater than 0.015 volts. In terms of volts, the adjustment should be in the opposite direction and of a value 1.3 times the measured change. This should produce a 0.7 times compensation, so that only a few adjustments should be needed to correct for a significant error.

Three digital input bits are used to furnish the rf "on" status, the bias supply "on" status, and the modulator "on" status. The bias supply program is one analog channel. The current waveform is the other one that is sampled twice to get the change. When the rf "on" status bit goes to the "off" state, the previous (15 Hz) cycle's reading of the bias supply current waveform is subtracted from the present value to get the change. As an example, if the observed change in the current waveform is 0.040 volts, an adjustment of 0.080 volts in the D/A is needed to correct it.

Data Capture Logic

Local Application

Wed, May 26, 1993

The CAPT local application can be used to assist in data capturing based upon a set of conditions that are indicated by Bit states. An “armed” Bit is turned off after a delay following detection of any trip condition. The armed Bit can be used to condition data capture via Data Access Table entries. The local applications parameters list for CAPT is as follows:

```
E LOC APPL PARAMS 05/26/93 1329
NODE<0576>  NTRY<10>
NAME=CAPT   CNTR=0193
TITL"CAPTURE DATA WITH DELAY "
SVAR=0004A73E
ENABLE  B<00BA>  CAPT ENABLE
ARMED   B<00BB>  CAPT ARMED
DELAY   C<0005>  CAPDL          CY
COND1   B<80BC>  CAPT COND1
COND2   B<00BD>  CAPT COND2
COND3   B<0000>
          <0000>
          <0000>
          <0000>
          <0000>
```

The enable Bit# is followed by the armed Bit#. When this bit is “1”, it is considered the “armed” state. Data collection via Data Access Table entries can be conditioned to execute when this bit is set. The delay Chan# reading specifies the number of 15 Hz cycles following the detection of a trip condition during which the armed Bit should remain set. The condition enable Bit#s are watched by the application to detect the occurrence of a trip condition. Any of the conditions is enough to be called a trip. The state of each Bit is indicated by the sign bit of the word. The remaining 15 bits specify the Bit# itself. In the example shown, when Bit# 00BC is a “1”, or when Bit# 00BD is a “0”, a trip condition has occurred. A word of 0000 means a “don’t care”. Thus, zero should not be used for a trip condition Bit#.

When CAPT is first enabled, it turns on the armed Bit. When CAPT is disabled, it turns off the armed Bit. When a trip is detected while in the armed state, the delay counter is set to count down. During the countdown, any further trips are ignored.

The initial program version is 220 lines of Pascal, generating about \$300 bytes of code. The maximum number of condition Bit#s can easily be increased (up to 7).

Delayed Reset After Trip

Data Access Table gymnastics

Fri, Oct 14, 1994

The problem:

Detect a trip condition via a status bit, wait for maybe a few minutes, and apply a reset control action to recover from the trip. This note gives a concrete example of one method to do this via Data Access Table entries.

Solution:

Build a counter to count cycles since the start of a trip condition. Use a period specification entry conditioned by the counter value to enable the following reset control action, plus a clear of the counter channel in case the reset is not immediate, in order to insure that subsequent resets cannot occur too soon.

Example:

Suppose in node 056B, operating at 10 Hz, status Bit 0198=0 indicates a trip condition, channel 0013 is used as the counter channel, a delay of 5 minutes is required following a trip before resetting, and pulsing Bit 011C high for one 10Hz cycle resets a trip. (Use two non-volatile memory words at 40FF70 for constants. Set up the counter channel as a dummy settable channel so the clear works.) The following Data Access Table entries follow the approach outlined above:

| | | | | |
|------|------|------|------|--|
| 1500 | 0013 | 0000 | 0000 | Build cycle counter in Chan 0013 when Bit 0198 transitions |
| 0000 | 8000 | 0198 | 0001 | to zero. Clear counter while Bit 0198 is one. |
| 7F00 | 0001 | 0000 | 0000 | Enable following entries when counter value falls <i>outside</i> |
| 0000 | C000 | 0013 | 0BB8 | range 0000-0BB8. (0BB8 = 5*60*10 cycles) |
| 0D95 | 0000 | 0040 | FF70 | Pulse Bit 011C high for one cycle. (40FF70)=0401. |
| 056B | 011C | 0000 | 0001 | |
| 0D81 | 0000 | 0040 | FF72 | Clear counter to limit rep-rate of reset. (40FF72)=0. |
| 056B | 0013 | 0000 | 0001 | (Do this in case reset doesn't set Bit 0198 right away.) |

If the reset pulse does not immediately cause Bit 0198 to become set, indicating a non-trip condition, then the last entry insures that the reset will only be done once. If the trip condition persists, and Bit 0198 continues to remain a zero, then the reset pulse will be re-issued 5 minutes later, when the counter again advances past BB8. If a reset occurs (manually) before the 5 minute timeout, the counter will be cleared.

DirectNET PLC Access

Local application

Thu, Feb 1, 1996

The vacuum controls interface for the PET project uses a Programmable Logic Controller to do the interlocks handling and vacuum-specific logic that is required. The IRM interfaces to the PLC via an RS-232 serial port. The basic approach is to routinely collect analog and digital data from the PLC, then map it into the IRM's analog and digital channels. Control actions are also output as necessary. All of this logic is handled by a local application.

The serial I/O input supported by the system software passes through the Serial Input Queue (SERIQ) table. By monitoring the contents of the SERIQ, all received characters of serial input, except the linefeed (0A) and null (00) characters can be seen. This is enough to catch the data coming from the PLC.

The DirectNET protocol can transmit data in hex (binary) or in Ascii. Although using Ascii requires twice the time for the data transfers, it helps to unambiguously detect control characters that are part of the protocol. This implementation of DirectNET support will use Ascii for that reason. The following control codes are used by the DirectNET protocol:

| | | | |
|-----|----|-----|----|
| ENQ | 05 | ETB | 17 |
| ACK | 06 | STX | 02 |
| NAK | 15 | ETX | 03 |
| SOH | 01 | EOT | 04 |

DirectNET Overview

One data transaction requires a series of I/O communications between the host computer and the slave PLC. To begin any transaction, the master sends an inquiry 3-byte sequence of "N", "address", ENQ, where address = \$20 + the PLC slave address. The slave responds with the same sequence, with the ENQ byte replaced by an ACK.

The master then sends a header that defines the operation. Its format is SOH, header, ETB, LRC. The LRC stands for a one byte Longitudinal Redundancy Check that is the exclusive OR of all the Ascii bytes within the header. The header itself consists of the one-byte (two Ascii characters) slave address, read (30) or write (38) character, data type character, two-byte starting address, one-byte #complete (256-character) blocks, one-byte number of bytes in last block, and one master ID byte (0 or 1). The slave responds with an ACK character.

For a read request, the slave continues by sending the data message in the format STX, data block, ETX, LRC. If the #characters in the data block is larger than 256, so that the #complete data blocks is nonzero, then more than one data block is sent, with each complete data block using a ETB character in place of the ETX. Each

data word within a data block is in byte order, least byte first. After each complete data block, or the last incomplete data block (of length 0–255 characters) is received by the master, the master returns an ACK. The slave then returns an EOT. The master finally sends an EOT. This final EOT clears the slave for future detection of an inquiry sequence.

For a write request, the transaction sequence is the same, but the data is transferred from the master and ACK'd by the slave. After the last data block is ACK'd by the slave, then the master sends an EOT to end the entire transaction.

Timeouts are imposed on the successive communications of a transaction. If a slave times out awaiting a response from a master, it will be necessary for the master to send an EOT to clear the slave to accept a new inquiry.

The details of this communication protocol are found in the manual.

IRM serial support

The usual serial port support in an IRM is organized around lines of input separated by a CR character. Nulls (00) and LF characters (0A) are removed from the input stream. If more than 128 characters are received without a CR, then one is inserted into the serial stream. By operating the DirectNET communications in Ascii mode, this should not cause a problem, as LF and nulls and CR aren't used. The slave sends a CR in Ascii mode. But a data block could be longer than 128 characters, so waiting for one would not be advisable. Therefore, as a first step, we will simply monitor (at 10Hz) what is found inside the SERIQ and in this way be able to see all characters in the stream as soon as they come in and are deposited into the SERIQ by the serial receive interrupt code.

Serial output support is usually organized as lines, with trailing blanks removed and CR and LF inserted. This is the usual way, but there is a separate listype that permits serial output without such editing. We shall use the latter listype for DirectNET output, in case the CR, LF would cause a problem for the slave PLC. The serial baud rate for use with the DirectNET interface is 19200 baud.

Data acquisition approach

The DNET local application program is used to collect the data routinely by sending a read transaction. The response data consists of two parts, the first for analog and the second for digital data. The response data is then mapped into the IRM's local analog channels and digital bytes. Between data acquisition transactions, DNET also monitors a message queue for setting commends, either to an analog word or a digital word. When a message is detected, a write transaction is made in place of the next data acquisition transaction. This approach means that all the support afforded analog channels and binary bits in the IRM system can be preserved. The acquisition may be slow, but this is not thought to be a problem for a

vacuum system controls interface. With this approach, an update rate of 1Hz or better, and a control action delay of less than one second, should be achievable.

In order to prevent other uses of the serial port for output, we may place a flag bit in the PRNTQ header that prevents such output. A simple way to do this may be to allow only the raw listype to work for serial output. Usual serial port output uses the normal output logic that edits out terminal blanks and adds CR,LF.

Message queue support

A change in the system code supports use of a PLCQ message queue. When a setting is made to a PLC-type device, a message about the setting is placed into the message queue. (If it has not been created, it will first be created.) In this way, there is a place for the settings that result for the Restore action following a system reset to reside, until the time that the DNET local application is initialized and the first data acquisition transaction completed. As DNET is initialized, it attaches to the PLCQ message queue so it can check for any waiting messages.

Parameters

Local application DNET parameters, using example test values, are as follows:

| | | |
|----------|--------|---|
| ENABLE | B 00D4 | Bit# enables local application |
| SLAVE | 0001 | Slave address of PLC interface |
| DATATYPE | 0001 | Data type# used for data pool acquisition |
| REFADDR | 1001 | Base reference addr for analog, digital data pool |
| NACHANS | 0010 | #chans of analog data |
| NDWORDS | 0008 | #words of digital data following analog data |
| MAPCHAN | C 0180 | Base analog Chan# for mapping to local IRM space |
| MAPBIT | B 0180 | Base binary Bit# for mapping to local IRM space |
| | 0000 | (spare) |
| | 0000 | (spare) |

The above set of parameter values supports 16 analog channels and 8 words (128 bits) of digital data.

Digital control scheme

Each BADDR entry is normally a memory address that should be written for the associated status byte. But 1553 and SRM communications required specially-coded 4-byte BADDR entries that are signaled by the use of hi byte values 80 and 81, respectively. For the PLC support, we use a hi byte value of 82. When the usual

binary data scan occurs, via the "0405" entry in the data access table, such entries are skipped. When a digital control setting is made, the data type and reference address are found in the lower three bytes of the BADDR entry. To perform the setting, the information must be passed to the DNET local application via the message queue scheme described above.

DNET collects the data pool from the PLC every 4 cycles. For support by a local application that is invoked at 10 Hz, this is the easiest approach. During the first cycle, the enquiry message is sent. On the second cycle, the 3-byte response to the enquiry is received, and the request header is sent. On the third cycle, the acknowledgment to the request header is received, followed by the data that was requested, and the ACK is sent. (If there is too much data, given the bandwidth available, then an additional cycle or more would be required.) On the fourth cycle, the EOT is received, and the EOT is sent to the PLC to clear it for receipt of the next enquiry. The message queue is checked for any settings to be performed. If one is found, then another four cycles is spent doing that write transaction. The required data type byte, reference address word, and data word are taken from the message queue entry, which was filled by the setting support in the system code using the contents of the BADDR entry. Upon completion of the write transaction, a new read transaction is performed that updates the data pool. As a result, the data pool is updated every 0.4 seconds, but when a setting must be performed, 0.4 seconds is taken to perform it. The maximum time between updates of the data pool is therefore 0.8 seconds. The maximum time to perform a setting, assuming none is already queued, is also 0.8 seconds. If a faster update rate is needed, a means of invoking the local application in response to serial activity will be required. At first, omitting such support is easier.

Bit-based, byte-based, and word-based digital control are supported. In any case, however, a word-wide setting is actually performed. Bit-based toggle, set hi, and set lo digital control types are supported. Bit-based pulse types are not supported for this hardware; the PLC's cpu logic can be used to do it.

Analog control

Analog control is specified by a new analog control type# \$19. The second byte gives the data type, and the last two bytes give the reference addresss to be used to effect the setting. It may be in the memory region that is part of the data pool, in which case the PLC's cpu will have to perform the setting to the real I/O module; or it may be in the I/O module itself. Upon successfully queuing the setting message, the setting word of the ADATA entry for that channel is updated, even though completely successful completion of the setting is not assured. Because knob control could queue settings faster than they can be delivered at 0.8 sec, the local

application checks for successive entries in the queue referencing the same target address (data type and reference address), and coalesces them as much as possible, delivering only the final setting it finds waiting in the queue.

Domain Name Server Access

Obtain IP address given node#

Fri, Feb 23, 1994

Each station has a node# that is used in data requests to indicate where a device resides. Historically, this node# has been part of the physical network address. But with the use of Internet Protocols, this natural node# is less directly useful. Given a node#, how can one determine the IP address needed to support IP communications?

Acnet developed a scheme that uses an alternate node# that is an index in a table of IP addresses that is downloaded to each computer that supports Acnet. As a result, each local station recognizes two node#s. One is the historically natural one, such as 0576; the other is the IP table index, such as 096F.

Until now, specifying a natural node# in a local station page application results in non-IP communications. Use of the 09xx node# forces IP communications. In order to obtain global access to local stations that are on different networks, especially common on ethernet, it may be desirable to force IP communications, if possible. If this is done, how can one derive the IP address given a natural node#? A convention has been used for this purpose in the host support of Macintosh and Sun computers. Rather than use the 09xx values, the natural node#s are used. The Domain Name Server is consulted to translate from a node name such as "node0576.fnal.gov" into an IP address.

Normal domain name resolver code operates synchronously; *i.e.*, it returns only after a reply is received in response to the query to the domain name server. In the context of a local station's real-time operation, this is inconvenient. To use DNS access, it will be necessary to operate asynchronously and to maintain a cache of node# vs IP addresses. If a name lookup query is sent to the name server, the frame due to be transmitted may be discarded. When a response from the name server arrives, the IP address can be placed into the cache, so that it will be available the next time access to that node# is needed.

Imagine a table with entries in the following format:

| | | |
|-------|-------|------------|
| node# | count | IP address |
|-------|-------|------------|

A nonzero node# means the entry is in use. A nonzero IP address means that an entry in use is complete. The counter is used to time out stale entries and also time out responses from the name server. Negative values of the counter mean that a request has been sent to the name server, but no reply has yet been received. Positive values mean that a countdown is in progress until the name server query is retried. If the counters "tick" at once per second, then the maximum delay (32K

seconds) is about 9 hours. An IP address of zero (with nonzero node#) means that no response has ever been received from the name server for that node#. If a time out occurs while awaiting a response from the domain name server, and the IP address is nonzero, then leave it the same, as the name server may only be temporarily unavailable due to network problems. If a response from the server indicates that the name is undefined, then clear the entry, as an IP address cannot be found for that node#. The table can also be maintained manually, in the absence of an domain name server, if desired.

When a node# is encountered that does not have an entry in the table, use a roving pointer to determine where to start a search for a vacant entry for the new node#. This insures that an entry will not be re-used too soon. In this way, one can use the entry#, perhaps plus an offset, for the request ID in the name server query message. The response ID will then indicate the entry whose IP address is to be updated.

This logic is implemented in a local application. Each LA is given a chance to execute each 15 Hz cycle, which provides the opportunity to perform timeout functions. The LA will also run, invoked by the SNAP Task, when a reply is received for the client UDP port, which provides for updating the appropriate table entry. The memory used for the table should be in non-volatile memory, so that the information is not lost upon system reset. As the system code needs access to the table also, the new IP Node Address Table (IPNAT) is system table #27. If the table is undefined, then this feature is be supported by the system. If the LA, called DNSQ, is not enabled, then the feature is supported, but only for node#s already in the table. In this way, one can implement support for access to a restricted set of nodes.

How does the system code that queues a message for a network via OUTPQX get the attention of the LA in order to carry out the job of issuing the name server query for an unfamiliar node#? When the LA is enabled, it writes its client *portId*, the entry# in the NETCT table of network and UDP port connections, into the IPNAT table header. The system code can check this *portId* value to lookup the message queue ID used for advising the LA of UDP network messages destined for its UDP port. Writing a special message to this queue advises the LA of the new node# for which it should send a new query. The system clears this *portId* word in the IPNAT header at reset time.

How can a system be configured to use only IP communications? In the third word of the PAGEM table is stored the "broadcast node#" used for name lookups and data requests that are addressed to multiple nodes using multicasting. If this value is in the UDP range of 09Fx, then we can assume the preferred use of IP communications.

IPNAT table header format

| | | | |
|--------------------------------------|----------|----------|---------|
| 'IN' | last | limit | start |
| portId | #retries | #queries | |
| DNS IP address | | #waiting | #active |
| | | | |
| up to 16-char suffix for DNS queries | | | |
| | | | |

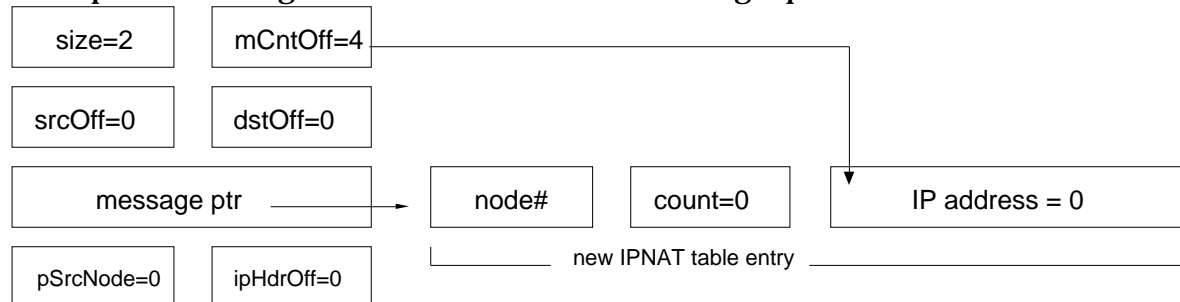
The 'IN' key serves to identify a valid table. (The IP default feature can be disabled by modifying this key.) The *last* word is the offset to the last entry used after searching for an empty one to install a new one. The *limit* word is the size of the table. The *start* word is the offset to the first entry. The *portId* is the index in the NETCT table, placed here by the DNSQ local application when it is enabled. The *#retries* is the count of the number of times that the LA had to repeat sending the name request to the DNS because of no response within the time out period. The *#queries* is the total number of DNS queries since reset. The IP address of the DNS is next, followed by the *#waiting* word, the #entries with waiting queries, and the *#active* word, the #entries actively awaiting a DNS response. A default *suffix* of up to 16 characters, blank fill, is appended to each node# name of the form "nodexxxx" with xxxx as the node# in hexadecimal. At Fermilab, the *suffix* ".fnal.gov" is used.

In the OUTPQX routine, when the destination node# is in the appropriate range to be a natural node#, currently 05xx, 06xx, and 07xx, and the system is configured to default to IP communications, *except for reply messages*, consult the entries in the IPNAT table to get an IP address, and call PSNIPARP to obtain a pseudo node# 6xxx. If no IP address is present, make a new entry for this node#, change the destination node# to zero so that it will be discarded, and write a special message into the message queue that is associated with the portID NETCT entry. This will invoke the DNSQ local application, which will see the special message and send a query to the name server for the given node#. (But it will return without waiting for a reply!) During subsequent 15 Hz LA processing, if no response is forthcoming, it will repeat the query. After perhaps 3 times, it will give up, setting the *count* word in the IPNAT entry to retry much later. In this way, responses from the DNS will be cached in the table.

Use of the *count* word for two purposes is as follows: It is divided into two bytes, cntHi and cntLo. Three cases depend upon the cntHi value. If cntHi > 0, cntLo is decremented each second. When it reaches zero, cntHi is decremented. If cntHi reaches zero, the time out has occurred that repeats the query to the DNS to catch

up on any IP address changes for that node. In the case that cntHi is negative, cntLo is also decremented each second. When it reaches zero, the query is retried, and the cntHi incremented. If cntHi reaches zero, it means that no response has been received from the DNS at all for this node#, and if the IP address is zero, the entry should be cleared, as the node# may be bogus. If cntHi = 0, no incrementing or decrementing is done. This allows for manual installation of table entries with static IP addresses. This is also useful for foreign nodes where the *suffix* does not apply.

The special message to be written into the message queue is as follows:



This special message is sent via the message queue to the DNSQ local application only when a new entry has just been placed in the IPNAT, the node# to be looked up has been placed into the entry, and the IP address field has been cleared. The *message ptr* is the address of the node# word in the new entry. The *msgOff* value of 4 means that the message count word is the hi word of the IP address field that has been set to zero. The logic in UDPRecv, called by UDPRead, that decrements the message count word to signal that a network message has been processed, first checks to see whether the message count word value is zero; if so, it does not change it. The other four words in this special message are not interpreted by UDPRead. Use of this special message format allows a UDPRead call in the LA to get the network replies from the DNS as well as the special message from the system notifying that a new node# needs to be looked up via the DNS. The LA identifies the special message format by the size=2, just enough for the new node#.

HUMBUG for Local Stations

Local application implementation

Dec 15, 1991

Yet another protocol generally supported by front end computers which make up the accelerator control system at Fermilab is HUMBUG. It provides access to the memory of a front end, as the data acquisition protocol does not support it.

Protocol

As implemented for the 68020-based local station front ends, two types of message formats are supported. The first is an Absolute Dump request whose format beyond the usual acnet header is as follows:

| Word | Meaning |
|------|------------------------------------|
| 0 | 1: Absolute Dump |
| 1 | #words of memory requested (1–128) |
| 2 | MSW of 32-bit address |
| 3 | LSW of 32-bit address |
| 4 | unused |
| 5 | unused |

The reply message format to this request is simply the data words requested. If the MLT bit is set in the first word of the acnet header—signifying multiple replies—the data is returned at 15 Hz until the request is canceled. If there is a bus error encountered, the status word in the acnet header will indicate it.

The second format is an Absolute Patch command, as follows:

| Word | Meaning |
|-------|---------------------------------|
| 0 | 3: Absolute Patch |
| 1 | #words of memory to set (1–128) |
| 2 | MSW of 32-bit address |
| 3 | LSW of 32-bit address |
| 4 | unused |
| 5 | unused |
| 6 ff. | Array of data words to set |

This command is always a one-shot. If the message type is a REQ, rather than a USM, a status-only reply will be sent. If there is a bus error, the acnet header status word will carry the news.

Klystron RF Gradient Regulation

Local application for Linac Upgrade

Thu, Nov 18, 1993

Introduction

The klystron RF systems drive the accelerating cavities in the Linac Upgrade. Due to temperature variations, the gradient in the cavities exhibits a variation over a period of minutes or hours. This note describes a local application that compensates for such variations to maintain more constant gradient readings.

The gradient amplitude affects the beam significantly. During beam cycles, the gradient is reduced by beam loading. Regulation of the gradient must concentrate on regulating *beam* cycles, using as a reference the nominal gradient value used by the alarm scanning system. On *no-beam* cycles, when no beam is accelerated, the program can measure the amount of beam loading by comparing the gradient reading against the last beam cycle reading. Then, after some time has passed without beam cycles, this beam loading estimate can be used to derive the appropriate reference for no-beam gradient regulation.

Parameters

The "Page-E" parameters are as follows:

```
E LOC APPL PARAMS 11/12/93 0835
NODE<0622>  NTRY<11>
NAME=KRFG   CNTR=0113
TITL"KLYSTRON RF GRADIENT REG"
SVAR=00033442
ENABLE  B<0090>  KFRG K2 ON/OFF
NBWAIT  C<0091>  K2WAIT      CYC
GRREAD  C<0484>  K2GRAD      NRM
GRSET   C<0422>  L2MSDA      V
GAIN    C<0090>  K2LPGN
AVGCYC  <0020>
LOADCYC <0080>
```

After the required enable Bit# parameter, NBWAIT is the Chan# whose value specifies the number of cycles after the last beam pulse at which time the program switches from beam regulation into no-beam regulation.

The gradient reading and setting Chan# are next. The gradient reading is in "normalized units," while the gradient setting is in "volts" units.

The relationship between a change in the setting in these units that produces a change in the reading units is about 0.25; *i.e.*, reading = 4 * setting. As a result, the next parameter, the gain Chan#, should be about 0.25 to correspond to full

correction. Note that significantly larger values of the gain may produce an oscillation.

The last two parameters specify periods in 15 Hz cycles. They are not Chan#'s, so their values must be changed via "Page-E" directly. The AVGCYC parameter specifies for the no-beam regulation mode the number of cycles over which readings are averaged to give a result that is compared with the expected no-beam reference value. It is probably easier to consider this as the minimum period between making no-beam adjustments. The LOADCYC parameter specifies the averaging parameter that is used in the computation of beam loading. The formula is as follows:

```
avgLoading:= (avgLoading*(loadCyc - 1) + loading)/loadCyc;
```

A new value of beam loading is measured on each no-beam cycle during the beam regulation mode, when there has been a "recent" beam pulse. This value of loading is combined in the above formula to produce an updated average value of beam loading. The average is used during no-beam regulation mode to derive the no-beam reference value for the gradient. The formula is as follows:

```
noBeamRef:= beamRef + avgLoading;
```

The beamRef is the nominal value of the gradient as used in the alarm scan.

Internal constants

Several internal constants are used by the program. To change them, one must modify the local application program called KRFG.

| | |
|-----------------------|--|
| minGrad = 0.75; | Minimum gradient reading for regulation |
| maxChange = 0.01; | Maximum setting adjustment made at once |
| maxSample = 30; | #cycles period for sampling parameter values |
| beamLoadStart = 0.04; | Initialized default value of beam loading |

Adjustment limits

If a calculated new setting is made that would take the setting channel outside of its alarm tolerance range, the setting is not performed. This serves to keep the range of adjustment within bounds, even if the gradient readings exhibit peculiar characteristics.

Local Applications Table

Support for closed loops, etc.

Aug 28, 1990

A means of expansion of the VME local station system software is by the use of local applications, which are separately-compiled procedures that are invoked by the system during Data Access Table processing. The Local Applications Table (LATBL) contains the entries that result in their invocation.

Invocation context

During Update Task processing of the Read Data Access Table (RDATA), one particular entry causes the LATBL entries to be processed. This means that local applications are invoked during updating of the datapool, likely to be positioned either at or near the end of RDATA.

Each local application (LA) is expected to be written as a Pascal procedure, just as are application page programs. The calling sequence is as follows:

```
TYPE
  TrigType = (init, term, kbint, cycle, net);
  ParamList = RECORD
    sVarPtr: sVarPtrType;
    enableBit: Integer;
    params: ARRAY[1..9] OF Integer;
  END;
PROCEDURE LocalApp(trig: TrigType; VAR LAEntry: ParamList);
```

The first argument is the same as that used by application page programs. (The `kbint` and `net` options may not apply.) The `init` call occurs the first time that the program is invoked since being enabled. The `term` call occurs when the LA is being disabled. The `cycle` call is the normal one given each 15 Hz cycle or whenever directed via a special Data Access Table entry.

The second argument is a ptr to a part of the LATBL entry. It points to a structure in the table entry reserved for a ptr to the LA's static variables and an array of up to 10 integers, the first of which specifies the local binary Bit used as the enable/disable control for this invocation. The other integer array elements may be anything else required by the particular LA.

During the `init` call, the LA is expected to allocate memory for its own static variable requirements. This can be done by calling this routine:

```
Function Alloc(sVarSize: Longint): sVarPtrType;
```

This call invokes the pSOS memory allocation routine and returns a ptr to the allocated memory. If the storage cannot be allocated, a NIL ptr is returned.

When the `sVarPtr` is returned by `Alloc`, the LA should save it in the first longword of its `ParamList` structure. This is necessary because the LA is a Pascal procedure that must be invoked multiple times during the time that its `LATBL` entry is enabled. Any information that must be saved by the LA across calls to it must be stored in this static variable space. Note that a given LA may be invoked multiple times with different `ParamList` structures. An example is the Linac rf gradient regulation that is to be done for 3 rf stations by one local station. This will use 3 entries in `LATBL` but only a single entry in `CODES` for the gradient regulation program (procedure).

When the `term` call is made, the LA should free its static variable allocation by calling this routine:

```
Procedure Free(statVarPtr);
```

This procedure simply frees the memory allocated by `Alloc`.

Local Applications Table

A new system table (#14) supports local applications. An entry in this table has the following format:

| | | | |
|-------------------------|-------|-------------|-----------------|
| status | count | name | |
| ptr to static variables | | enable Bit# | other params... |
| | | | |
| | | | |

The status word is a copy of the previous enable bit reading. Comparing this value with the current enable bit reading allows the system logic to decide what to use for the `trig` argument in the call to the LA. The enable bit, when set, signifies that the entry in `LATBL` is enabled. When an LA entry makes a transition from disabled to enabled, the `init` call is used. The LA is expected to allocate

during this execution, the act of disabling a local application means it will “start over” when it is re-enabled.

The program that is to be run is identified by 4-character name. Along with the type code of `LOOP`, the `CODES` table of downloaded separately-compiled programs is searched for a match, and the address of the executable copy of the program is used as the target for the call. The first time that the program is accessed, for the `init` call, a checksum check is performed to insure that the downloaded code has not been corrupted, and the program is copied into newly-allocated dynamic memory for execution. This means that the downloadable area is always available to receive a new version.

Downloading a new LA

When a change is to be made in a LA program, the new code is downloaded without concern for the currently-executing code. The LA scan finds the name of the LA and the ptr to the executing code (in on-board memory) in the `CODES` table entry corresponding to that name. The process of downloading leaves this pointer alone while the code is copied into a newly-allocated area.

When downloading is complete, the checksum is sent to be stored in the `CODES` table entry, and the ptr to the downloaded code is marked (by setting its `ls` bit) to show that it is a fresh copy.

LATBL table processing

When `LATBL` entries are scanned by the Update Task, and a fresh downloaded copy of the code is detected, and the type of call was to be a `cycle` call, the call type is altered to a `term` call. This gives a chance for the LA to free any allocated static variable storage and “clean up its act” in general. After any `term` call, the saved copy of the LA’s enable bit reading is cleared. This will cause an `init` call to be given on the next cycle if the LA is still enabled. The checksum will again be checked and new memory allocated for execution in on-board ram.

After all `LATBL` entries have been scanned, a separate scan is made over the `LOOP` entries in the `CODES` table. For each entry which has the fresh download bit set, the bit is cleared, the executable area is freed and its ptr cleared.

The result of the above logic is that those entries which use the program just downloaded will be disabled and re-enabled automatically the very next cycle. (If it is desired to prevent an alarm message from being sent, in the case that the enable/disable status bit is being monitored, one can merely elect to use the `2X` option with that status bit, since the bit will be disabled for only one cycle.) This means the new version of the code will take effect right away. To prevent this, either disable all `LATBL` processing by disabling the Data Access Table entry, or

The index page logic directs the call-up of application pages. When a page is being called up, if the lo byte of the longword which contains the pointer to the entry point of the application page program is nonzero, the 4-byte “pointer” is assumed to be a program name of type `PAGE`. (Note that this implies that using a ptr in the old way can still work as long as the entry point address is on a 256-byte boundary.) A search is made for a match in the `CODES` table, and the download area is sum-checked, memory is allocated in on-board ram for it, and the program is copied into that area for execution.

When the program terminates, either because a new page is called up or a return is made to the index page, a scan is made of the `CODES` table. The allocated area of any `PAGE` type entry in the `CODES` table is freed, and its pointer is cleared. This is done because only one `PAGE` program can run at a time. Note that this is in contrast to the `LOOP` type programs, in which many can be running at once.

TASK or INIT processing

New tasks may be added to the system code by making a scan at reset time which looks through the `CODES` table for any entries of type `TASK` or `INIT`. Such entries can be copied into executable memory and called. What they do depends upon how they are written. Such a program could spawn and activate a task, for example. Or it could simply do some job at reset time. Only one call would be made to such a program, and it would be made from the `ROOT` task. This adds another dimension to system configuration possibilities.

Local Control Box

Booster High Level RF

Tue, May 28, 1996

R. Goodwin, B. Peters, R. Florian

As part of the new installation of controls for the Booster high level RF system, a local control box was proposed to replace the front panel of the MIU crate, used since about 1970 to interface I/O signals to the Lockheed Electronics MAC-16 mini-computer. (The MIU interface depended heavily on the particular I/O channel interface design supported by the MAC-16 and is unavailable for use with a replacement system.) The control box purchased for the new local control support is a DynaComp GreyLine 2200 Series operator panel that provides a four-line 20-character alpha numeric display, a numeric keypad that includes 6 additional keys, a row of 8 function keys, and a set of 8 labeled LEDs. The labels for the keys can be configured by the implementer. The box interfaces to the IRM serial port at rates up to 19200 baud.

| | | | | | | | | | | | | | | | |
|-----------------------|-----------------|------------------|--|------------------|---|-------------|-----|------------|--|-------|--|-----|--|------|--|
| FB12O | | | | 7.452< | | | | V | | | | | | | |
| MD12V | | | | 24.83 | | | | KV | | | | | | | |
| PT12SI- | | | | 0.001 | | | | A | | | | | | | |
| RF12GE | | | | 54.39 | | | | KV | | | | | | | |
| <input type="radio"/> | FBS OFF | | | 7 | 8 | 9 | Inc | | | | | | | | |
| <input type="radio"/> | Modulator OFF | | | 4 | 5 | 6 | Dec | | | | | | | | |
| <input type="radio"/> | Cavity Short IN | | | 1 | 2 | 3 | +/- | | | | | | | | |
| <input type="radio"/> | Spark Detect | | | Clr | 0 | Ent | Set | | | | | | | | |
| <input type="radio"/> | spare | | | | | | | | | | | | | | |
| <input type="radio"/> | spare | | | | | | | | | | | | | | |
| <input type="radio"/> | Volts Display | | | | | | | | | | | | | | |
| <input type="radio"/> | D/A Display | | | | | | | | | | | | | | |
| FBS ON OFF | | MOD ON OFF | | CAV IN OUT | | Trip Log | | A/D D/A | | Volts | | Sel | | Page | |

Booster High Level RF

After some local testing and consultation with representatives of the high level RF group, several suggestions seemed attractive. The display itself is used to show up to four parameter lines. Each 20-character line provides for a 6-character name, a space, a 7-character numeric value field, a control marker character, a space, and a 4-character units text field.

The Clr button on the numeric keypad is used as a "clear entry" when entering a numeric value. The Ent button is used to commit to an entered value and perform the setting. The +/- button allows entering an arbitrary signed decimal setting value. The Inc and Dec buttons permit incremental adjustment for an analog setting. In either case, the setting targets the parameter indicated by the control marker.

The labeled lights on the left side show whether the FBS or Modulator is OFF, whether the Cavity Short is IN, whether a spark was detected on the last update cycle, and whether volts and/or setting values are being displayed.

The bottom row of push buttons toggles between ON/OFF or IN/OUT states. The Trip Log button shows the summary trip counts plus the time of the last clearing of trip counts. Press the Trip log button again to return to the normal four-parameter list.

| | | |
|-------|----------|------|
| TRIPS | 11/09/95 | 0758 |
| FBS= | 10 | |
| MOD= | 3 | |
| STA= | 16 | |

The A/D D/A button toggles between displaying reading values and setting values on the four-line parameter list. The D/A display light indicates when setting values are displayed. This mode is indicated by an engineering units field of "V . ". The Volts button causes the display values to be in A/D (or D/A) volts units. Press the Volts button again to revert back to normal engineering units display.

The Sel button sequences the control marker through the controllable parameters of the current available set of four-parameters, in case more than one such parameter is controllable. The Page button sequences through the available four-parameter displays. If the control box isn't used for a period of time, it will revert to the first four-parameter list.

The functionality described above is supported via a local application called HLRF that was written by Bob Peters. See the following URL for more information:

http://garlic.fnal.gov/booster_controls/

Local Station Additions for Acnet

Local applications and more

Dec 26, 1991

The Linac controls upgrade to the local station software required several additions to that software in order to fit into the Acnet system requirements. This is an *ad hoc* list of some of those changes.

Local applications

A local application is a software procedure that is invoked by the system software to provide a particular feature without being linked into the system code itself. It is therefore a modular addition to the local station software. Any local application can be independently enabled and updated to a new version even while the system is running. It is known by name and not by memory address; space for a copy in non-volatile memory is handled automatically by the system support. A given program can also be updated in all local stations at once using token ring network group addressing. Examples of existing local applications and their purposes follows:

GRAD

Regulates rf amplitudes in the cavities to compensate for long term drifts

PHAS

Regulates rf intertank phase for long term drifts

CROB

Automatic recovery from pa crobar trips of rf system

DRIV

Automatic recovery from rf system driver trips

PINH

Automatic reduction in rf gradient setpoint after rf trip w/o recovery

QUAD

Automatic recovery of drift tube quad power supply trip

PRES

Regulates ion source pressure

Note: the above set of 7 closed loops always existed in Linac local stations.

AERS

Alarm Event Reporting System shepherds alarm messages to AEOLUS, the Acnet alarm-handling process which runs on some Vax.

FTPM

Fast Time Plot Manager supports requests for data to be used by the Fast Time Plotting facility which runs on Vax consoles.

AAUX

Acnet Aux provides for "ping"-type support to check communications.

GATE

Permits gateway support of Acnet-header communications allowing for copying memory from an SRM to any other SRM.

NETM

Network Monitor ensures that network frame reception is working and automatically re-connects to the network if it is lost.

TEMP

Regulates water system temperature for the new Linac rf modules

KRMP

Assists during conditioning of the new rf modules. Also collects detailed spark statistics.

FREQ

Controls a VCO to keep the cavity resonant following application of a large change in rf power to the cavity.

Page applications

Application display programs that run on the small local consoles have always been a part of the local station software. The original Linac supported the standard set of 4 below. Several new page applications have been added in the course of the controls upgrade for Linac.

PARM

The parameter page is used widely in Fermilab accelerator controls systems to support general display, control and plotting of analog parameters.

EDAD

Edit Analog Descriptors provides for entry of *analog* device information into the local station database.

EDBD

Edit Binary Descriptors provides for entry of *binary* device information into the local station database.

MDMP

Memory Dump allows flexible inspection of memory for both hardware and software diagnostics.

Note: The above 4 are the original basic suite of page applications.

SURV

The Survey page assists in maintenance of the large number of token ring nodes that are attached to the token ring network.

REQR

Request Reply timing can be histogrammed for the Classic protocol.

CRTI

CRT Image permits running a page application remotely on another node. Implemented in another platform, it permits running any page application from another platform, such as a workstation or Macintosh.

SRMC

This SRM Copy page uses the GATE gateway local application to support the copying of memory between any pair of SRMs.

New modules

Several new modules provide additional support needed for the new Linac controls system.

ACREQ

This large module provides complete support for the accelerator data acquisition protocol, sometimes referred to as RETDAT/SETDAT. Included is support for offset/length to provide generic access to the local station database to assist in uploading to the central database. Also included, to reduce network traffic to the consoles, is server support, in which all Linac device data is retrieved from a Vax console through a single "server node", which in turn collects more efficiently from the "real" source nodes. Again, to reduce the usual network traffic load on Vax consoles using the standard Parameter Page, averaging support is provided for analog device readings, preferentially averaging beam pulses over no-beam pulses.

OPENPRO

To facilitate the support for additional protocols, a set of routines is used with a Protocol table to route communications for more protocols through the ACREQ module. In particular, it is now possible to write a local application that supports a new network protocol of the Acnet-header type. Examples of such LAs are FTPM, AAUX, and GATE.

ARCINT, SRMREQ

Arcnet network support was added to the system in order to communicate with the SRMs, which are the data acquisition interface for the new Linac. A new simple data acquisition protocol was designed for use with the SRMs. Special data access table entry types were also needed to permit collection of data via Arcnet, taking maximal advantage of arcnet communications.

LOCAPPL, SETPROG

These two modules provide the support for local applications and the associated downloading of same.

ASTATUS

Composite status words are assembled from the digital I/O interface via a table-driven scheme. These words can be scanned for alarms in parallel, to mimic the traditional Tevatron notion of power supply status words.

Network Frame Monitor

Local application

Wed, Jun 4, 1997

The FMON local application is designed to aid network frame diagnostics. It can capture a part of network frames received or transmitted to/from a selected target node, or from all nodes. It also checks for duplicate frames by monitoring the *identification* field in the IP header.

Traditional embedded network diagnostic

In each IRM or local station system, network frame diagnostics are included that write into the NETFRAME data stream (DSTRM entry #0) a short record of each network frame that is received or transmitted by that node. The record contains the node#, the size of the frame, a ptr to the frame buffer holding the frame, and the calendar time of the frame reception or transmission, with resolution down to half milliseconds since the start of the current 15 Hz cycle. A page application NETF allows viewing of the contents of this data stream. It can also allow viewing frame contents, but this part can only work if one asks to view a given frame before the circular buffer of frames is overwritten by successive receive or transmit frame activity. For busy systems, especially including IRMs that connect to ethernet, one may have only a few seconds before the buffers are overwritten. This makes it very difficult to do some kinds of frame diagnostics using this embedded tool, when it is necessary to view frame *contents* at leisure.

New method of capturing frames

The FMON local application runs in a system for which frame diagnostics are needed for frames associated with communication of that node with another node. The approach in using FMON is that of doing an experiment, rather than operating a typical utility. These are the FMON parameters available on Page E:

```
E LOC APPL PARAMS 06/03/97 1438
NODE<0509>  NTRY<28>/64  H<0508>
NAME=FMON  CNTR=47  DT= 0.5  MS
TITL"NET FRAME MONITOR      "
SVAR=00044D9A      05/06/97 0931
ENABLE  B<00EF>*FMON ENABLE
NODE#   <E071>
ON      B<00EE>*FMON CAPTURE ON
MAX INDX <2000>
DIAGPHI <0030>
DIAGPLO <0000>
W OFFSET <0011>
FLAGS TR <0003>
```

First there is the usual enable Bit# needed for each local application instance. The node# parameter identifies the target node for which frame communications are of interest. In the case of IP-based communications, this is a pseudo node# that

corresponds to an IP socket, the combination of an IP address and a UDP port#. The pseudo node# uses 6 or E for the hi hex digit, where 6 is used for token ring and E is used for ethernet. The middle two digits identify a 16-byte entry 02-FF in the IPARP table, and the least digit is a port# index that refers to a UDP port# via the associated port# block that is allocated for use while an IPARP table entry is active. An entry times out after no activity is observed for about an hour. Port# index values of 1-F signify valid UDP ports in current use. An index value of 0 means a port# is not used, such as for the case of an ICMP ping request/reply message or an IP datagram fragment. So, if a pseudo node# is used as a target node# parameter, it really refers to a node and port combination. In order to find out what pseudo node# value to use here, one may have to check the traditional NETFRAME diagnostics during a trial run.

The ON capture Bit# parameter allows enabling or disabling the frame capture logic without disabling the local application itself. This is convenient because one needs to check the FMON static variables—allocated dynamically when the local application enable bit is set—to get information about where the frames are currently being captured. The maximum index parameter refers to the size of the memory available for capturing frame contents. A 64-byte data structure is used for frame capture; thus, only a portion of the frame may be captured. In this example, the maximum index parameter of \$2000 means that there is room in memory to capture parts of 8K frames. The two following parameters together comprise the 32-bit base address of memory that is to be used for the frame capture data. For IRMs it is common to use \$300000-3FFFFFF for this purpose. This is the fourth megabyte of memory on the CPU board, currently not used for system purposes.

The word offset parameter allows capturing parts of frames not beginning with the start of the frame header. For example, for ethernet, an offset value of \$07 will skip the 14-byte frame header. For IP frames, a value of \$11 will skip both the frame header and the 20-byte IP header. A value of \$15 would additionally skip the 8-byte UDP header, allowing capture of frame contents beginning with the start of the UDP datagram contents. the captured data is actually limited to 56 bytes, as the first 8 bytes of the 64-byte record is for the calendar time of the frame reception or transmission. The "TR FLAG" parameter refers to whether receive, transmit, or both receive and transmit frames are to be captured. The least significant two bits are used for this purpose:

- | | |
|---|---------------------------|
| 0 | none |
| 1 | receive |
| 2 | transmit |
| 3 | both receive and transmit |

Note that the use of a 64-byte structure to hold captured frame contents is a

convenient match to the Memory Dump page application—not an accident. One can advance through successive frames using that diagnostic page application and very quickly identify changed fields of interest.

The FMON program builds on the traditional frame NETFRAME diagnostics in order to provide its functions. It uses a local data request to monitor the records written into that data stream each 15Hz cycle. When it gets a match for capture, it uses the buffer ptr to grab the frame contents that are to be captured, during that same 15 Hz cycle, thus insuring that the network buffers will not be overwritten before the contents can be captured. The capture circular buffer is typically quite large; if necessary, the capture enable bit can be turned off to assure that captured frames may be examined at leisure until the cows come home.

Duplicate frames

By monitoring the *identification* field—designed for identifying IP datagram fragments that belong to the same IP datagram in order to allow IP fragment reassembly to work—in the IP header of received frames, FMON can notice when duplicate frames occur by maintaining a small table of such data for frames recently received from the target node. This logic was installed at a time when we were experiencing such problems that turned out to stem from overloaded network hardware. There is no guarantee that a particular IP implementation would be compatible with this specific duplicate frame logic, but those that were tested implemented the standard by using an incrementing counter value for every IP datagram transmitted, a scheme that insures there will be no repeat until 65536 datagrams have been transmitted. It's simple and it works.

Method of operation

To perform an experiment in frame capture diagnostics, prepare suitable parameter values that will be needed, using Page E. The parameter that will likely require the most attention is the target node#. For IP frames, this will be a pseudo node# word that represents a UDP socket, as described earlier. A trial run may be needed to determine this value; use the NETFRAME diagnostics via Page F to get a handle on this. It can capture the occurrences of the last 237 frames received or transmitted. One can usually identify the frames of interest by the frame size and period of activity, so that one normally doesn't have to be quick enough to capture the actual frame contents using that program. Enter the node# thus determined as the target node# parameter value. It will be good for an hour or so, at least, because that is the timeout for ARP table entries. When that much time passes without any IP receive or transmit activity for that node, the ARP table entry will be cleared. The next time access occurs with the same node, it may quite possibly be assigned a different table entry than before, and will therefore have a different pseudo node#.

Choose the word offset used for frame capture according to what part of the frame contents needs to be captured. Select whether receive or transmit frames should be captured. Enable the capture flag bit. If it is already on, toggle it off and back on. This is optional, but it will cause the capture to begin at the start of the memory used for the purpose. The memory stores frames in a circular buffer mode; therefore, the last "MAX INDX" frames will be captured and available anyway, but it may be more convenient to know that the capturing starts at the beginning. When the capture enable bit transitions from 0 to 1, the internal logic resets the capture circular buffer pointer to the beginning.

Perform the experiment during which frames must be captured. If it is necessary to also capture frames relating to a second target node#, then install a second instance of the FMON local application. Obviously, different parameters must be used for the enable bit, although the capture enable Bit# parameter could be the same for both instances. But be careful not to use the same area of memory for the capture circular buffers needed by each instance!

When the condition is reached for which the frame data should be studied, one can turn off the capture enable Bit#. This will freeze the capturing. In some cases, this won't be necessary, as the condition that is reached may terminate the communication anyway. As an example, this tool was used to capture frame data so as to learn what caused snapshot plot activity to terminate. The condition reached was cancellation of the snapshot request and therefore termination of communication activity; no further frames would be captured anyway. It depends upon the exact nature of the experiment being performed.

To examine the frames captured, one can observe some of the static variables in use for this local application. Using the ptr shown on Page E, one can see the following picture on the Memory Dump Page for the time in which this was captured as an example:

```

8 MEMORY DUMP      06/04/97 1055
0509:044D9A 0904 0000 0000 0007
0509:044DA2 0509 0091 0000 BF0B
0509:044DAA 01CE 2000 0000 01CD
0509:044DB2 0030 0000 0030 7300  Ptr into capture circular buffer
0509:044DBA 0001 0001 0001 0001
0509:044DC2 0101 0000 0000 0000
0509:044DCA 0000 0000 0000 0000
0509:044DD2 0000 0000 0000 0000

0509:044DDA 000E 000E 0000 0000
0509:044DE2 0000 0000 0000 0000
0509:044DEA 4F06 BE76 2FE8 A059
0509:044DF2 12CB 823A F126 DC96
0509:044DFA AD8E C98E E08E F88E
0509:044E02 1E8F 3B8F 558F 6E8F
0509:044E0A 948F AC8F C48F DD8F
0509:044E12 0390 568E 6E8E 878E

```

On the first "page," fourth line, in this example, the address 00307300 is the pointer to a 64-byte structure containing the date and up to 56 bytes of the contents of the last frame captured. Here is that structure:

```

0509:307300 9706 0410 5506 0083  Time 6/4/97 10:55:06 cycle 0, 33ms
0509:307308 008A 008A 00E6 DADE  UDP header (srcPort=dstPort=138)
0509:307310 1102 12BD 83E1 7BD9  UDP datagram contents used for
                                NETBIOS datagram protocol
0509:307318 008A 00D0 0000 2045
0509:307320 4246 4145 4945 4A45
0509:307328 4543 4143 4143 4143
0509:307330 4143 4143 4143 4143
0509:307338 4143 4143 4141 4100

```

Since the word offset parameter was \$11 for this example, the captured frame data begins at the UDP header, assuming IP communications are used. In this example, the UDP port# used is 138. According to the RFC document that describes Internet Protocol Assigned Numbers, this port# is used for NETBIOS Datagram Service. The UDP length is 230 bytes. It was nothing of local interest, but in this example, pseudo node# E071 just happened to correspond to aphid.fnal.gov, presumably a PC. The last experiment using FMON in node0509 was done many days ago, after which the meaning of the pseudo node# changed to another node. Still, it serves here as an example. One can find the significance of the pseudo node# by looking up the entry in the IPARP table, currently based at \$40E000 in all IRMs. At \$40E070 we find the following ARP table entry:

```

0509:40E070 00A0 2491 A138 0000
0509:40E078 83E1 7BD9 0004 3A98

```

The port# block at \$43A98 is as follows:

```
0509:043A98 0060 0426 0000 0015 Port# memory block is type $15.
0509:043AA0 002C 0003 0EBE 03CE
0509:043AA8 0000 0000 0000 0000
0509:043AB0 0000 0000 0000 0000
0509:043AB8 0000 0000 008A 0000 Each entry in array is port#, use count
0509:043AC0 0089 0000 0000 0000
0509:043AC8 0000 0000 0000 0000
0509:043AD0 0000 0000 0000 0000
0509:043AD8 0000 0000 0000 0000
0509:043AE0 0000 0000 0000 0000
0509:043AE8 0000 0000 0000 0000
0509:043AF0 0000 0000 0000 0000
```

In the port array, the port index 1 (lo 4 bits of \$E071) corresponds to port 138. The IP address is 83E17BD9, or 131.225.123.217, which is `aphid.fnal.gov`. Looking at previous captured frames, it appears that this frame, presumably broadcast from `aphid.fnal.gov`, is received every 15 minutes. If this level of activity continues, this ARP table entry will not time out.

Duplicate frame diagnostics are shown on the second "page" of the first memory display. The two values of 000E followed by 0000 indicate that the recent time-stamp values and corresponding *identification* field values from the IP header. (The 000E values are indexes into the arrays of time-stamp and *identification* field values.) The fourth word of 0000 indicates that no duplicate frames received from the target node were tallied during the time since the capture enable bit was last set to 1. The time-stamp array is a byte array starting on the third line; the *identification* field value array begins on the fifth line.

Network Time Protocol Client

It's about TIME!

Thu, Apr 21, 1994

Local stations have always maintained correct time-of-day, derived from an NBS satellite clock receiver that transmits via RS-232 a 13-character Ascii message that is interpreted by one of the stations on the network, which then broadcasts the result to the rest. To use this system in another place, one would need to purchase such a radio receiver. But with the widespread use of Internet, another scheme taking advantage of the Network Time Protocol to access a server seems an easier means to get the time-of-day. This note describes an implementation of an NTP client as a local application. It periodically queries an NTP server, interprets the reply, and updates the local time-of-day. Between queries, the time-of-day is maintained via timing based upon a crystal on the CPU board.

The NTP protocol is a simple client server model. A client sends a simple request addressed to UDP port# 123 of an NTP server node. The server returns a reply that includes an 8-byte time-of-day that consists of a 4-byte integer number of seconds and a 4-byte fraction. The time value is based upon January 1, 1900. Since some time in 1968, the value of this 8-byte quantity has been negative; *i.e.*, the number of seconds since 1900 has been more than 2^{31} . The value will “roll over” in 2036. In the meantime, we should be able to talk about time-of-day reasonably accurately.

The parameters used by this local application are as follows:

(put picture here)

Every local application has an enable bit as the first parameter. The second parameter for the TIME local application is the period between queries expressed in seconds. (For example, 003C = 60 seconds.) The next two word parameters comprise the IP address of the NTP server to be queried. Here, 83E17C28 is 131.225.124.40, the IP address of CNS33, a Fermilab NTP server. Next is the number of retries before timing out. Next is the time zone correction in hours relative to GMT, which depends upon whether Daylight Savings Time is in effect. Finally is given the multicast target node# for sharing this news with other local stations on the network.

Now for the details. The Network Time Protocol is described in RFC-1305 in 300K of detail. A much easier treatment is in RFC-1361, describing the Simple Network Time Protocol that is suitable for end user clients. SNTP uses the NTP in a simple way. This SNTP variation is used by the TIME local application.

The message format is 15 longwords, or 60 bytes, in length. In the SNTP variation query format, the first byte is \$0B, with all other bytes zero. The meaning of this first byte is that the Version Number = 1, and the Mode = client. Version 3 of NTP is described in RFC-1305, Version 2 in RFC-1119, and Version 1 in RFC-1059. There was a Version 0 described in RFC-959 that is no longer supported by NTP servers. An NTP server that supports Version “n” also supports lower versions down to 1.

The reply to such a query is up to 60 bytes in length, where the 8-byte Transmit Timestamp is extracted for interpretation as the present time-of-day. It is found 40 bytes deep into the message. To convert this form into the time-of-day requires some work. The TIME local application may not be the best example of how to do it. But here is a description of its current implementation:

Empirically, it has been determined that the value of seconds since the start of 1994 is \$B0CF3B80, or 2966371200. This value assumes the Greenwich Mean Time zone, which is 6 hours ahead of Central Standard Time. The software subtracts this 1994 base from the Timestamp value and derives the rest of the month, day, hour, minute and second from the rest. If more than a year has passed since 1994 (January 1, 1995, or later), it assumes 365 or 366 days of seconds per year. Our calendar system is not quite perfect, so that the time-of-day in some years has an extra “leap second” inserted after the last day of the year. The Timestamp returned by NTP servers, however, does *not* include any such “leap second,” so the time-of-day produced by TIME should be ok for subsequent years. (This is explained in RFC-1305.)

Once the conversion has been made, the local record of the time-of-day, in BCD at memory address 00000788, is set to the current time. Then, if the target node parameter is nonzero, a message is prepared using Classic Protocol to send it to the local stations on the same network. The address that is used for this depends upon a table of transmittable multicast addresses in the TRING table in non-volatile memory. This table of 8-byte entries is at address 00105B80 in 133-based local stations and at address 00405B80 in 162-based stations and Internet Rack Monitors.

A timeout of 2 seconds is allowed before assuming no response from a server. This is a constant in the TIME local application. The number of retries parameter should allow for a timely response very soon, however, in case routers need to perform ARPs, say, and fail to pass an unexpected frame the first time.

NonLinear Units Conversion

Local application

Fri, May 28, 1993

Radiation monitors called “scarecrows” and “chipmunks” are used to measure radiation in beam enclosures at Fermilab. The readouts are non-linearly related to radiation units of mr/hour. This note describes an easy implementation for support of the linearization of such signals, in order to make it easy for arbitrary host platforms to scale to engineering units linearly.

An example of the setup of this RADC local application is as follows:

```
E LOC APPL PARAMS 05/28/93 1036
NODE<0576>  NTRY<11>
NAME=RADC  CNTR=00AE
TITL"RAD MONITOR CONVERSION  "
SVAR=0004942C
ENABLE  B<00BE>  RADC ENABLE
FORMULA1 <0001>
INPUT1  C<0000>  N576T0      V
OUTPUT1 C<000B>  RADT1      MRH
FORMULA2 <0002>
INPUT2  C<0000>  N576T0      V
OUTPUT2 C<000C>  RADT2      MRH
FORMULA3 <0000>
INPUT3  C<0000>  N576T0      V
OUTPUT3 C<0000>  N576T0      V
```

Up to three conversions can be specified in each instance of this local application. Each specification includes the formula index#, the raw channel#, and the result channel#. The formula index#s are 1 for chipmunks and 2 for scarecrows. The input channel provides the raw voltage reading. The output channel is a dummy channel whose scale factors have been suitably chosen to fit the expected range of the result and the resolution needed. Ultimately, it becomes a 16-bit signed value.

The operation of the logic is to get the reading of the input channel, apply the indicated formula, and write the result to the output channel using engineering units. The formulas used are of this form:

$$\text{radiation} := \text{Exp}(c0 + v*(c1 + v*(c2 + v*(c3 + v*c4))))$$

The two cases supported simply use different coefficient values. Additional formulae could of course be added. This local application isn't restricted to radiation conversion. Any constants and coefficients needed would be part of the program code, however.

The source code is 183 lines of Pascal, executing in less than 1K bytes.

Resistor-Temperature Conversion

Tesla cryogenic system local application

Wed, May 11, 1994

The TRTD local application supports needs of low temperature measurements for the Tesla superconducting RF laboratory in LAB-2 in the Fermilab village. Four signals are demultiplexed into 16, thermal emf's measured and used to get the resistance of the cold resistors, which are of two types, platinum and carbon. From the resistance, calibration tables are used with linear interpolation to get the temperature in degrees Kelvin. This note describes the methods used to do this.

Parameters layout

The available parameters for TRTD as seen via "Page E" are as follows:

(put picture #1 here)

The usual enable Bit# argument is followed by MPXDATA, the base channel# of the four multiplexed signals. V⁺, ETC is the base channel# of the sixteen V⁺, V⁻, emf, resistance, and temperature values, altogether 5 sequential "arrays" of 16 channels. The R REF reference resistor channel# specifies the Pt reference resistor. The next channel# holds the Carbon reference resistor value. MPXSEL is the base Bit# of the digital byte used for selecting the four 1-of-4 multiplexer values. The program sequences this byte through values of 00,AA,55,FF on each four successive cycles. The CURDIR parameter is the base Bit# of the two bits used for controlling the current direction through the Pt and C resistors, respectively. The EMF PER parameter is the number of cycles between measurements of the V⁻ and emf values, done by reversing the current direction for four cycles to collect these data.

Demultiplexing

Signals from the RTD system are very small, so they are amplified by four-input multiplexed amplifiers. Two-bit select codes are wired to each amplifier. The program operates all four two-bit select codes at once, as they share the same byte of digital output. Data is read from one selection each cycle, which for the LAB-2 facility is 10 Hz. In this way, all 16 channels of data that result are updated every 400 ms.

Thermal emf's

To measure the thermal emf's that result from the conductor leads that connect from room temperature to cold temperatures, it is necessary to reverse the current. Every so often, perhaps every few minutes, the current is reversed and the data collected for the negative direction of current flow. After taking four cycles to accomplish this, the current is returned to the usual positive direction. The formula for the emf is as follows:

$$\text{emf} = (V^+ + V^-)/2$$

These emf values are updated every time a new V⁻ is measured with the current direction negative.

Resistances

The resistors used are of two types, platinum and carbon. The single Pt resistor calibration table is fairly linear down to liquid nitrogen temperatures, so linear interpolation can be used with a set of 10 calibration points only. These Pt calibration points are part of the program source and are as follows:

(put picture #2 here)

The formula for resistance is as follows:

$$R = ((V^+ - \text{emf}) / V_{\text{ref}}^+) * R_{\text{ref}}$$

Here V_{ref}^+ is the voltage across the reference resistor (Pt or C) and R_{ref} the reference resistor value itself. The reference values are constants in the program. The values at the time of this writing are $R_{\text{ref}} = 99.82$ (Pt) and 1000.0 (C).

The carbon resistors calibration curve is very nonlinear, so more calibration points are used. Linear interpolation is done with 20 calibration points between the *log* of the temperature and the *log* of the resistance. In order to collect these calibration data for use by the TRTD program, the calibration data points were entered into Excel by editing the original full calibration data set text file. In the spreadsheet, it was easy to calculate an average of the 5 calibrated carbon resistors for use with those carbon resistors that have no measured calibration data. In addition, logs were calculated for the temperature and resistance calibration data points. These log values were output from Excel via a text file, which was then edited into an MPW assembly source file containing “DC.S” data directives. This source file was then assembled and the resulting data downloaded into the station that will run the TRTD local application. Upon initialization, the program requests this local data file and places the data into its calibration data arrays for use in converting the carbon resistances into temperatures. Part of the calibration data entered into Excel was this:

(put picture #3 here)

The calculated logs of the calibration point data are as follows:

(put picture #4 here)

These data were then edited as data declaration statements into an MPW assembly source file, assembled and downloaded as a data file of 32-bit floating point values called DATATRDT into the local station along with the LOOPTRTD local application. When the program is first initialized, it reads the data file for use during Carbon resistor temperature conversion.

Internal details

Four channels are read each cycle and converted as necessary. Of the 16 channels read, they are of the following “resistor types”:

(put picture #5 here)

The resistor type specifies which reference resistor is used as well as which calibration data to use to perform the temperature conversion.

During program operation, a data structure allocated at initialization keeps track of the context. Using Page E, one can find the location of this dynamically assigned data structure. Its structure at the time of this writing, expressed in the format produced by the memory dump page application, at a time when the signal values may not have been valid, is as follows:

(put picture #6 here)

The diagnostic `deltaTimes` byte array shows elapsed times per 10 Hz cycle in units of 0.5 ms. (A value of 02 means 1 ms.) Since the program operates on 4 signals per cycle, these times exhibit a periodicity of four cycles.

All floating point values are in IEEE standard 32-bit single precision.

Debug mode

A special debug mode is available for checking the ohms to temperature conversion for the Carbon resistors. To use it, find the above structure using Page E. Then set the `debug#`, which is initialized to zero, to the index value 1–16 of the channel whose conversion is to be checked. When this word is nonzero in this range, the program stops doing its usual work and concentrates on only the given channel index. It accepts the resistance value from the resistance channel, rather than computing a new one based upon the current demultiplexed data readings. This allows a value to be set into a resistance channel and the temperature channel observed resulting from this conversion. One can make a calibration plot of temperature versus resistance in this way using, for example, the Parameter Page on the Macintosh written by Bob Peters. Here is an example of making such a calibration curve for the T6 signal:

(put picture #7 here)

Logarithms, etc

Because of the extremely nonlinear characteristic of the resistance of the Carbon resistors relative to temperature, it was necessary to work with logarithms of these parameters. With linear interpolation on the *logarithms*, reasonably accurate results can be obtained. But the IRMs use a 68040 cpu, which does not have support for logarithms and powers on-chip, although it does have support for the basic add, subtract, multiply and divide. So a routine was needed to compute logarithms and powers. In a book called “Approximations for Digital Computers” by Cecil Hastings, Jr., Princeton University Press, 1955, can be found suitable formulae for these and other functions to several degrees of accuracy. The formula for $\text{Log}_{10}(x)$ that was used in TRTD has an error function < 0.000002 over the range $1 \leq x \leq 10$. Its form is as follows:

(put picture #8 here)

RET/SETDAT Test Page

Local Station Application

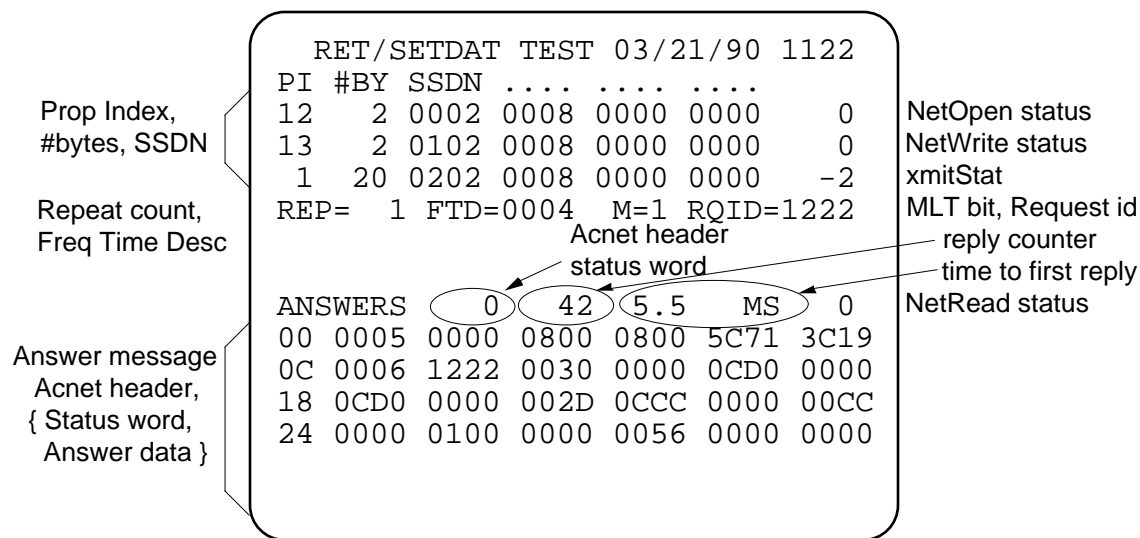
Mar 22, 1990

Introduction

A standard message protocol has been used since 1982 for accelerator control at Fermilab between the console computers and the front end computers. It is referred to locally as “DAS,” (Data Acquisition Services) or as “RETDAT/SETDAT,” after the network task names that are assigned to receive data requests and settings, respectively.

The Linac control system interfaced to this protocol via the Linac front end PDP-11, which translated between this protocol and the “classic” protocol that the Local Stations have always supported. The new Linac control system Local Stations will use this standard DAS protocol directly. The VME local stations now support this message format for data requests and for settings. This application page exercises this protocol by issuing data requests or settings to a local station and displaying the responses that are returned. It is written using the local station’s Network Layer interface routines.

Display page layout—data request mode



This snapshot of the test program display serves as an example of several of its features. It shows an example of requesting a 2-byte reading, a 2-byte setting, and a 20-byte analog alarm block all from channel 0 in node 08. The FTD specification indicates that the data is to be collected at intervals of 15 Hz (66 msec).

Parameter entry for data requests

The M bit is set for periodic requests. The request remains active until a cancel message is sent. If the M bit is zero, the request is automatically cancelled after

update of the first set of reply data. The `RQID` is the request id that serves to uniquely identify one request from another.

Enter up to three device specifications. The Property Index and the #bytes requested/device are specified in decimal. The SSDN (SubSystem Device Number) is entered in hex. It is a 4-word field that is normally extracted from the central database given a property index and a device index. (The reference that describes these terms more fully is Acnet Design Note 22.28.) To omit a device spec, blank out the property index field.

The Property Index values of interest are:

- 1Analog alarm block
- 3Basic Control
- 4Basic Status
- 5Digital alarm block
- 7Extended status
- 12Reading
- 13Setting

The formats for the 4-word SSDN used in Linac are as follows:

| | | | | |
|------------|---------|----|----------|---------|
| listype# | idLng=2 | or | listype# | idLng=3 |
| node# | | | node# | |
| Chan#/Bit# | | | memory | |
| — | | | address | |

These examples give the long ident forms for channels, bits and addresses. The short form is also supported for backwards compatibility, where the `idLng=1` word for channels and bits, and `idLng=2` words for addresses.

The Frequency Time Descriptor (`FTD`) expresses the repetition period in 60 Hz cycles. The `REP` is the repeat count for testing the timing for longer requests. It merely repeats the same set devices to build a request with up to $3 \times \text{REP}$ request packets, each of which requires 16 bytes in the network data request. Thus, if you enter a count of 100 for a single device spec, the request message is more than 1600 bytes in length.

Interrupt anywhere in the parameter specification area of the screen (on rows 2–8) to initiate the data request. The word `ANSWERS` in the response area of the screen should be hi-lited to indicate the request is active. (If it is a one-shot request, or if an error is returned, it may not stay hi-lited for long.) When the first (or only) response is received, the elapsed time is displayed on the `ANSWERS` line in milliseconds. This time is measured from just *before* the call to `NetWrite` until

just after the call to `NetRead` that returns the first response in the test program.

Just to the left of the elapsed time to first response is the decimal count of replies received. The Acnet header status word is also shown in decimal just after the word `ANSWERS`. This is done for convenience in interpreting error codes, which are negative numbers.

Other status replies that are shown are toward the right end of the screen. The status return from `NetOpen`, which is called upon entry to the page, is given at the end of the third line. Below that is the return from the call to `NetWrite` when the request message was issued. Below that is the transmit status word that gives the success of the network transmission. At the end of the `ANSWERS` line is the status return from the call to `NetRead` which returns the reply data.

Cancelling data requests

Cancel an active request by an interrupt on the `ANSWERS` line. The hi-lighting will be removed as the cancel message is sent (a USM with bit#9 set in the Acnet header flags word). (Note that an interrupt on this line when a request is *not* active switches to the setting mode.) Leaving the page results in all active requests being cancelled the next time a reply is received. This happens because the application closes its network connection (via `NetClose`) upon exit.

Answer data viewing

Six lines are used to display answer data in hex with 6 words per line. The byte value at the left shows the offset in bytes of the first word on the line. There are three ways to adjust the starting offset for the block of answers.

The easiest way is to use the raise/lower buttons on the local console. It will adjust the offset by 72 bytes (6 lines of 6 words each) at a time. If you advance too far, it wraps to the start.

To adjust the offset so that the first word is one of the displayed data words, merely interrupt under the word you want to be the starting word displayed. Obviously, you can only move forward in this way.

The third way to adjust the offset is by typing in the desired offset in the first characters of the first line of answer data and interrupting. Three characters can be entered, even though only the least significant two characters can be displayed due to the screen's limited number of characters per line.

The entire response message is displayed, beginning with the 9-word Acnet header. (For an error response, this is all you get.) This is followed by a reply packet for each device, consisting of an error status return code followed by the answer data for that device, padded to an even number of bytes.

Details of the example response

The first 9 words are the Acnet header. The first word shows that it is a reply message with the `MLT` bit set. The next word is the reply status word, and it is also shown in decimal on the line above. The destination and source bode of the request are both node 08 in this case. This example illustrates use of the network to make a request to itself. (If it didn't do that, it would always require two nodes to do the test.) A by-product of this is that the `xmitStat` value shown at the end of the fifth line is -2, indicating "address not recognized," which in this case is normal.

The next two words of the Acnet header are the destination task name, which for Acnet data requests is `RETDAT`, in the Radix-50 encoding commonly used by PDP-11 computers. The source task id is 6, which denotes the table index in the Network Connect Table returned by the call to `NetOpen`. The request id is followed by the message size word, which is the total size of the response message including the Acnet header itself.

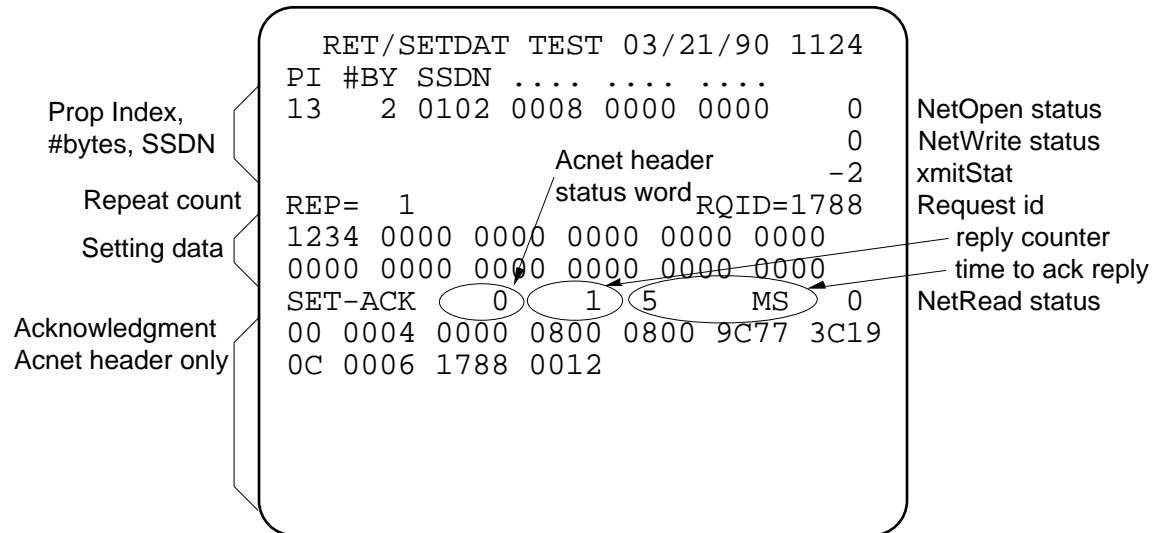
The Acnet header is followed by the reply packets. In each case, the error status word is zero, indicating no errors. Both the reading and setting of the device are `0CD0`. The analog alarm block includes the nominal and tolerance values of `0CCC` and `00CC`, respectively. The value `0056` is the trip counter.

The time response value for this example is 5.5 msec. This seems to be a common value for a one-shot minimal data request in any of the three protocols supported by the Local Station software. As of the time the snapshot was taken, there had been 42 replies received.

Switch between request/setting modes

Interrupt on the ANSWERS line to switch into setting mode from request mode. Interrupt on the SET-ACK line (same one) to switch back to request mode from setting mode.

Display page layout—setting mode



Parameter entry for settings

The device spec data is entered the same way as for requests. In addition, the setting data may be entered. There are 24 bytes of setting data available. If more setting data is required to satisfy the device specs, the setting data words are simply repeated. When the REP (repeat count) is more than one, the setting data is re-used starting at the beginning for each repetition. No FTD value nor M bit value is used for settings.

Setting acknowledgment

The reply to a setting is the status-only acknowledgment message. It consists only of the Acnet header with the status code in the second word. For the example shown, The 0004 indicates a reply messages, the destination node and source nodes are both node 08 as before, the destination task name was SETDAT in Radix-50 encoding, the source task id was 6, the request id was 1788, and the total length of the reply message is 18 bytes. The time to respond was 5 msec.

Serial Server

Local Station application

Oct 17, 1989

Overview

The Serial Server facility for the Fermilab D0 VME systems provides for a serial RS-232 connection to the Token Ring network of VME stations. A computer plugged into the serial port of any VME station can request data from any devices in the entire network of stations. In addition, settings can be made to any device in the network. These two commands are supported with the full generality inherent in the VME station software, the only real limitation being that of the serial baud rate. One can program any application in a familiar environment and yet have access to all the data in the system.

The Serial Server is implemented as an application program running on a VME station. The station used does not have to be the same station providing the serial port, although that might be a likely. Since it is an application, however, the server functions will stop if one leaves the application page. Currently there is a limitation of six simultaneously active requests supported by the server. A request for data can be a one-shot request, or it can be repetitive. The maximum repetition rate is 15 Hz, although one may reach the baud rate limit soon at that rate if too much data is being requested. The serial data is returned via a spooling mechanism using the available dynamic memory, so that the full baud rate can be delivered.

Message encoding

The command messages used to request data and make settings are of the same format as is used by the Token Ring network, but it is expressed in an ASCII encoding method based upon the Motorola S-record object format. That format provides for an S0 header record, followed by any number of S1 data records comprising the content of the block, followed by an S9 record which terminates the block. Each record includes a length byte, a two-byte "Load Address" (LA), any content bytes, and a checksum byte, all expressed in hexadecimal ASCII at two characters per binary byte of data. (Note that hex digits A-F are expressed in upper case only.)

The S0 header record LA is used in a *response* to indicate an error when nonzero. The header content data bytes allow the user system to tag the request with a request-id, which may be used to match the returned data block with the request. The S9 terminating record uses the load address field to specify the number of S1 data records included in the block. The receiver can test this count to insure that no data records were lost in the block transmission. In order to use a format which is *incompatible* with the Motorola format, the non-hex character S is expressed as a lower case s. This is to insure that such serial records do not mistakenly get used as input to the Download application program.

Examples

An example of a request for a single reading of channel 04:000 (channel 0 in system 4) to be returned at a 1 Hz rate is as follows: (Note that the spacing indicated here is to aid the eye for illustration only.)

```
s0 05 0000 0001 xx<CR>
s1 0D 0000 2001 0F 01 2001 0002 0400 xx<CR>
s9 03 0001 FB<CR>
```

The first record specifies a request-id of 0001—it may be any value up to 10 bytes long. The second record comprises the entire request command. The 2001 identifies the command as a request-for-data using list 1, the 0F is the count of 15 Hz cycles representing 1 Hz repetition rate, the 01 is the number of listtypes (which specify the type of data requested), the 2001 specifies 2 bytes per ident and only one ident in the request, the 00 is listtype 0 (A/D readings), the 02 means that two bytes are requested, and the 0400 is the (short) ident for channel 0 of system 4. The third record includes the 0001 to indicate the count of s1 records in the block.

The response to be expected from this command might be as follows:

```
s0 05 0000 0001 xx<CR>
s1 09 0000 0001 0000 4000 xx<CR>
s9 03 0001 FB<CR>
```

The header record specifies 0000, signifying no errors were detected in the previous request command. (If the request command was received and recognized, but something was wrong with it, the reply would have consisted only of an s0 record with a nonzero LA field.) The 0001 echoes the request-id sent in the request command. The s1 record has a 0001 to indicate that it is an answer message (the hi nibble=0), and that the list number =1. The next 0000 word means no errors from the data collection system. The word of 4000 is a hypothetical reading value, which would represent here one-half of full-scale, or 5 volts at the A/D input. The s9 record again provides the s1 record count.

In almost all of the above records, an xx is shown to signify the checksum byte. It is computed by summing the binary bytes (not the ASCII character codes) beginning with the record length byte and ending with the last data byte, and one's complementing the result to form the byte which is then ASCII-encoded. (As an example, the checksum for the s9 record above is FB.) Notice that the receiver when checking for checksum errors merely has to sum all the bytes mentioned above plus the received checksum byte. If the result is FF, the checksum is correct.

An example of a setting command might be to set channel 7 in system 4 to one volt. The setting command would look like this:

```
s0 05 0000 0056 xx<CR>
s1 0B 0000 3001 01 02 0407 0CCC xx<CR>
s9 03 0001 xx<CR>
```

The 0056 value in the s0 record is the arbitrary request-id. The s1 record shows a 3001, which signifies a setting command, with an ident length of one word. The 01 byte is the setting listtype #, and the 02 means we are sending two bytes of data. The 0407, of course, represents the channel ident (short form) specifying channel 7 in system 4. The data value is 0CCC, which is one-tenth of full scale, or 1 volt. The s9 record is as before.

Serial Server Application

Here is the application page format of the Serial Server as displayed on the small consoles of the VME systems:

```

S SERIAL SERVER      01/31/87 1332
INPUT SYS:2          #RECORDS=    21
*ACTIVE              #ERRS=       0
  L  REQS      NB  ANSWERS      NB  E
*1    12      36    1230      16
  2     1    144         1    308
  3
  4
  5
  6

S00500000003xx
S10D000020020001200100020700xx      data request
S9030001FB
S1090000000200005678xx              response
```

(This sample is purely illustrative and does not represent completely consistent example data values.)

Upon entry to the page, the second line indicates *ACTIVE. To change the node whose serial port is used for the serial server, enter the system number of the station which is attached to the serial port (if changed), and with the cursor on the 2nd or 3rd line, press the interrupt button. The second line will indicate *ACTIVE as above and will remain so until one leaves the page, which will terminate operation of the server. The rest of the page merely shows diagnostics of the server operation.

The number of records received from the user system and the number of error records detected is shown. The table shows diagnostic values for each of the six possible active requests which can be handled by the server. The * before the request number indicates that the request is currently active and collecting and delivering answers. The second

field shows the number of separate request commands received using the given list number. The next field is the number of bytes (not ASCII characters) comprising the request itself. The next field counts the answer blocks returned, and the next one is the number of bytes in the returned answers. The last field shows any error status of the data collection system.

The last lines display a snapshot of the latest request that was received. The first 3 show the header record, the first data record, and the terminating record of the request. The last record shows the first data record of the most recent answer response, which will continue to update until a new request is received, when these 4 lines will change to give diagnostics for that request.

Baud Rate Timing

To get an idea of what the baud rate timing limitation is, assume that 9600 baud is used, and we request the single word of data as described in the first example. The number of ASCII characters in the answer response is 52, assuming that <CR> represents both a carriage return and a linefeed character. At about 1 ms per byte, this amounts to 52 ms of serial time—quite busy at a 15 Hz rate but rather comfortable at 1 Hz. Further, if we requested a set of 20 readings—again originating from any stations in the network—the time would be about 140 ms, assuming that the response block contained two s1 records. And if we requested 100 words, it would require about 500 ms. Of course, operation at 19.2K baud would halve these figures. Therefore, we have that the serial time for n data words=

(48+4.6n) ms @ 9600 baud

(24+2.3n) ms @ 19.2K baud

Settings Log Implementation

Device setting activity

Thu, Sep 8, 1994

Due to the open access available in the local station/IRM system, it is useful to build a log of recent setting actions as another diagnostic tool for analyzing system behavior.

A *data stream* approach has been used to implement a local settings log. The advantage to this approach is the support already built in for data stream access. For some time, a data stream (#0) has been defined to hold recent network frame activity. A host, wishing to monitor recent network frame activity, uses a periodic request for the data stream contents, processing each new entry it receives. Multiple hosts can monitor the queue without interference—a feature of data streams.

The settings log, using data stream #1, is designed to work in a similar way. The network frame diagnostic page application called NETF was used as a start to build a page application for viewing the setting log data stream contents. The information recorded is the node# of the node issuing the setting message from the network, the listype#, #bytes of setting data, the channel#, the data value, and the time-of-day the setting was successfully performed, organized as follows:

(put picture #1 here)

Settings log for Acnet

In the Acnet system, it is desired to have a log of setting activities directed to Acnet devices. In case of some problem with the accelerator, this Acnet settings log can be a useful diagnostic tool for helping to determine the cause of the problem. Vax console applications have been logging settings they originate via the library routine that provides Acnet setting support for user application programs. Other sources of setting activities have been included in this scheme over time. Since local stations support a small screen display program that includes many attributes of the usual parameter page, and since Linac technicians *do* use this tool in working with Linac hardware, it is important to also bring such applications into the Acnet settings log scheme, referred to as “settings accountability.”

The data stream described above houses entries that describe settings performed at the target front end level, as opposed to settings initiated at a console level. To fit into the Acnet scheme, it is necessary to record settings that are initiated by relevant applications that run in the local station/IRMs. *This is a very different type of settings log than that described above.* For this reason, code has been added to the local station’s version of both the parameter and analog descriptor pages to record a special entry in the same data stream. This special entry includes the device name and property mask. By sharing the same data stream, we can get space for the name and property mask by eliminating the time field, which is not used in the Acnet scheme anyway. The adjusted record format is as follows:

(put picture #2 here)

The Key byte has a value that distinguishes this record format from that used for local setting activities, in which this byte of the date/time field would be a value in the range \$00–\$14, denoting the BCD 15Hz cycle counter.

With these special entries placed into the data stream, a local application has been written that monitors the contents of the data stream for these special entries and builds device setting accountability records for Acnet, delivering them according to the specified protocol for that purpose, designed and implemented by Kevin Cahill. See the document *Settings Log LA* for more details on this local application.

Acnet device names are 8 characters in length, such as L:GR2MID for the mid-tank gradient reading for RF system #2. The local stations use only the last 6 characters in their device names. The two-character prefix is supplied by a parameter to the local application running in that station. To cover the case of a setting that targets a device with a different prefix that is normal for the station running the application, the settings log server, upon finding no device entry in the Acnet database, performs a search for a device that matches the last 6 characters to correct the record.

Settings Log LA

Settings report for Acnet

Wed, Sep 7, 1994

Introduction

This document describes a local application that acts as an interface between the settings log data stream and the Settings Log Server implemented in Acnet via a task running on node `cfss.fnal.gov` that accepts a special protocol via a UDP port# used for that purpose. Setting records of a special format are written into this data stream from page application that run in a local station/IRM.

LA parameters

Example parameters as shown on Page E for the SLOG LA are as follows:

| | | | |
|-----------|---------|-------------|---|
| ENABLE | B<00C4> | SLOG ENABLE | <i>Enable Bit#</i> |
| DB PREFIX | <4C3A> | | <i>'L:'</i> |
| IPADR HI | <83E1> | | <i>cfss.fnal.gov IP address</i> |
| IPADR LO | <797A> | | <i>131.225.121.122</i> |
| RPT DLY | <000A> | | <i>10 seconds report delay</i> |
| REM DLY | <0258> | | <i>600 seconds removal delay</i> |
| LISTNUM | <000C> | | <i>list# for data stream data request</i> |

The `DB PREFIX` parameter is a two-character ascii prefix to all 6-character device names for preparing the settings report message that must include the 8-character device names used in Acnet. For example, the Linac device name prefix is "L:". Parameters `IPADR HI` and `IPADR LO` specify the IP address for node `cfss.fnal.gov`. This node houses the settings log server to which all settings reports are sent for logging. The `RPT DLY` parameter is the "report delay" in seconds. It specifies the delay after a setting is made before a report will be sent. Its purpose is to allow a chance for the setting activity of other devices to be queued for inclusion in a single report message. The `REM DLY` parameter is the "removal delay" in seconds, after which time the queued entry is removed. During this time, additional settings of the same property made to the same device will not be re-reported. Its purpose is to reduce network traffic resulting from "knobbing" an analog device. The `LISTNUM` parameter is used for the internal data request that monitors the contents of the settings log data stream.

Settings reporting

Upon initialization, the SLOG LA allocates a settings queue into which setting log activity that relate to Acnet device names are queued for eventual inclusion in settings log reports. When a setting is observed by the LA in monitoring the records in the settings log data stream, its listype# is checked for those of interest to Acnet.

This table lists these listype#s:

| |
|-------------------------|
| Analog setting property |
| 1 setting |

- 7 relative "knob" setting
- 39 delta setting
- 41 engineering units setting
- 44 engineering units delta setting

Basic control property

- 22 digital control via Chan

Analog alarm block

- 2 nominal value
- 3 tolerance value
- 4 alarm flags w/o \$4000 bit
- 42 engineering units nominal
- 43 engineering units tolerance
- 57 D0 alarm parameters

Digital alarm block

- 2 nominal value
- 3 mask
- 4 alarm flags w/ \$4000 bit

A settings log queue entry is as follows:

```

RECORD
dName: PACKED ARRAY[1..6] OF Char; { device name }
sMask: Byte; { set properties mask }
rMask: Byte; { reported properties mask }
remov: Integer; { time out until entry removed from queue }
nRprt: Integer; { number of reports }
nSets: Longint; { number of settings to this device }
END;
```

The setting records monitored are only those that are sent from an application such as the parameter page or the analog descriptor page, as only such records include a device name. Assuming that the queue is empty, and a setting of one of the above listypes is detected, then get the analog name from the channel descriptor. If it is valid, then build a new entry in the queue. Set a bit in the `sMask` field according to the setting property used. Set the `rMask` field to zero, as no properties have yet been reported. Set the `remov` field to the `REM DLY` parameter. Set the global report delay timer `rDTimer` to the `RPT DLY` parameter.

If a setting is encountered for a device whose entry is already in the queue, set the appropriate bit in the `sMask` field, then check whether `sMask = rMask`. If they are equal, then this property has already been reported to cfss in the last removal delay period,

so this one can be ignored. If they are not equal, and the rDTimer is zero, then set it to the `RPT DLY` parameter to insure prompt reporting of the new-property setting.

Every second, decrement the rDTimer, if nonzero. When zero is reached, review all queue entries and build a report message that includes all unreported properties of all devices in the queue.

Every second, also decrement the `remov` field in each entry. If it reaches zero, and `rMask = sMask`, all properties have been reported, so remove the entry from the queue. If it reaches zero, and `rMask <> sMask`, then some properties for this device have *not* yet been reported, so reset the `remov` field to twice the `RPT DLY` parameter. The rDTimer should already be active in this case. This extra time should give a chance for the unreported properties to be reported before the entry is removed.

When issuing a report message, the LA uses the acknowledgment typecode option for the cfss-based server. This causes the server to return an acknowledgment, which is used to set the rMask bits according to the mask that was reported. If there is no acknowledgment received within a second, then re-issue the report up to two times before giving up.

Device name prefix

The prefix used for all Linac devices is “L:”. Devices at the MRRF test station are “Z:”. From the local station application point-of-view, the correct prefix cannot be known. So, for all reports issued by a given station, a parameter of the LA is used as a prefix. In cases where a target device is in another node whose devices use a different prefix, the report will indicate the wrong device name. For a device with a name which the setting log server cannot find in the Acnet database, the server will perform a search throughout the database looking for a match on the last 6 characters of the device name in order to complete the setting log record.

Station identification

Each station has a 16-character text title that usually shows where it is located. For example, node 0508 has the title “LINAC SUN ROOM”. In the settings log protocol, there is an 80-character owner field for each report message. Besides the node#, the node’s title is included, normally identifying the local station’s location.

TFTP Implementation

Trivial File Transfer Protocol Server

Tue, May 17, 1994

This note describes the support for the TFTP (Trivial File Transfer Protocol) for the local stations. Local stations do not have mass storage, but they have non-volatile memory that is used for the storage of local and page applications. It is therefore useful to consider this UDP-based standard protocol as a way to distribute application programs to the local stations. A motivation for doing this, rather than the present way of downloading new programs via the serial port, is that it makes it easier to access files from host machines, as modern hosts support this standard protocol. Besides, it's faster.

There are two sides to the support needed: client and server. The client side, if implemented for the local station, would be a page application. Its implementation on a Macintosh, as recently done by Bob Peters, is an MPW tool. The client allows a user to specify a program name and a target IP address. For named programs, the name is an 8-character string, such as LOOPGRAD, that gives both the 4-character type name as well as the 4-character program name of that type. The client initiates either a read or a write transfer. The server side accepts the read/write transfer that is initiated from a client. In addition, support is included for access to the system code of a local station, or more generally, for access to an arbitrary block of memory, thus providing a kind of cheap save/restore facility.

Client logic

A read transfer copies a named program, say, from the target IP address to the initiating node, whereas a write transfer copies a named program from the initiating node to a target IP address. Blocks of 512 bytes are sent, receiving an acknowledgment for each block, until less than 512 bytes remain, when 0-511 are sent in the final block.

For a write transfer, the size of the file is known locally, so it is easy to source the file to the target IP address. Blocks are sent 512 bytes at a time, until the final one that is from 0-511 bytes in length tells the receiver (server) that it is the last. Each block is acknowledged by the server. If no acknowledgment is received, then the block is resent. Check sums for named programs are handled by the server, as the protocol does not support them. Acknowledgments are considered enough to make the transfer itself reliable.

Server logic

The server is a local application that supports both read and write requests. A read request received by the server means that the server should transmit the program to the client. Again, the server has an easy time sourcing such requests, since the size of the program is known locally. If no acknowledgment is received, the server re-transmits the last block sent. After the final block is sent, the server activity may linger regarding that transfer to allow for the fact that the acknowledgment might not be received for the last block. Only after the acknowledgment has been received, or the server times out the

transfer, should the server “close the connection.”

A write request received by the server means that a program is being copied to the server node. This means that the transferred file must be read into temporary memory at first. At the conclusion of the transfer, the size is known, so the appropriate settings can be made to build the resulting program file into the non-volatile memory. The solution to this problem is to use dynamic memory to allocate each received block of up to 512 bytes. When the last block has been received, settings can be made from each allocated block of data and the block freed.

Memory access, System code “file”

A filename of the form “MWxxxxxxxx.ssssss” can be used to access memory data. The xxxxxxxx is the starting address and ssssss is the number of bytes of memory to copy. MW accesses the memory as words, MB as bytes, and ML as longwords. The specified address can be less than eight hexadecimal digits, and the size can be less than the six digits shown. The special name “SYSTEM” can be used to access the system code “file” as memory words. This name can be installed in 162bug’s network boot parameters for use with the automatic network boot procedure following system reset.

Uses

Besides booting the system code via 162bug, named program access may be used to build an archive of such programs on a workstation’s disk. The memory access can allow save/restore of blocks of non-volatile memory. The main use for the write support in the server is to allow sending system updates to a development target node during program development and debugging. The usual download page can be used to disseminate this code to other local stations, individually or via multicast addressing. Note that time-stamping of named programs is done by the server upon reception, so if a new version is only targeted to a single test node, and then disseminated from there to other nodes, the time-stamp can serve as a “version date.”

Performance

The 162bug can accomplish the network boot function of its startup operation in about 2 seconds, a small part of the total of approximately 20 seconds required. The Macintosh client can transfer the system code in about 5 seconds to the TFTP server local application called LOOPTFTP. As a replacement for use of a serial port to download a new version of the system code, even at 19200 baud in 4 minutes, it’s great. In addition, the S-record text file does not have to be generated. For named programs, the time is also much improved, and the S-record file generation is no longer needed.

Data request “filename” option (not implemented)

Consider support of a general data request protocol layered on top of TFTP. To keep the filename short, use only single listype/single ident requests. Also, to simplify the problem, consider only one-shot requests. The filename might have the following format: `LTxx-xxxx-xxxx.ssss`. The `xx` fields would specify the listype, the source node, and the ident in hex. The `ssss` could specify the requested number of bytes. (The fields could have a different number of hex digits than shown here.) When such a filename is received, the server would make a data request. When a reply is received, it would return the data. In case the number of bytes requested is more than 512 bytes, a temporary array will be needed to hold the data. Therefore, a limit of one such active request could be supported at one time. Note that a request for less than 512 bytes of data will result in a single reply datagram that will be acknowledged exactly once. Since a host will write a file every time such a response is received, this may not work at a high rate. But at least it can capture data using only a standard protocol.

Vacuum Units Conversion

Local application

Mon, Sep 20, 1993

Ion pump power supply currents are used to measure vacuum in the cavities of the Linac upgrade accelerator modules. Vacuum readings in Torr are non-linear with respect to the readings of the ion pump power supply currents. This note describes a simple implementation for support of the non-linear conversion, so that hosts can scale the vacuum readings linearly.

The format of the parameters to specify for this units conversion is as follows:

```
E LOC APPL PARAMS 09/20/93 2129
NODE<0627>  NTRY<12>
NAME=VACT   CNTR=0520
TITL"VACUUM TORR CONVERSION  "
SVAR=00035B22
ENABLE  B<00B2>  VACT-2  ENABLE
FORMULA1 <0002>
INPUT1  C<01C2>  V1VPAB      V
OUTPUT1 C<0362>  V1VTAB      -9T
NCHANS1 <0006>
FORMULA2 <0002>
INPUT2  C<01D0>  V4VPAB      V
OUTPUT2 C<0368>  V4VTAB      -9T
NCHANS2 <0008>
        <0000>
```

Up to two conversion sets can be specified for each instance of this local application, called VACT. Each specification gives the formula index#, the initial input channel#, the initial output channel#, and the number of sequential channels to be so converted. The output channels are dummy channels whose 16-bit reading words are updated to reflect the results of the calculations. The scale factors for these channels are chosen to suitably fit the expected range of the result with the needed resolution. Since vacuum pressure tends to vary over a wide range, a compromise may need to be made.

In the initial implementation, the scale factors of the dummy channels used to hold the linearized readings are $fs=3276.8$ and $off=3276.8$. This gives a range of 0–6553.6 in units of 10^{-9} Torr, or a maximum value of 6.5×10^{-6} Torr. The least bit resolution in the 16-bit word is 10^{-10} Torr.

The formulas used to fit the calibration data for the ion pump power supplies are as follows, where v is the input reading in volts:

| Formula index# | Formula | Used for: |
|----------------|---|-------------------------|
| 1 | <p>If $v \geq cThr$ then</p> <p style="padding-left: 20px;">$vacuum := s * \exp(c10 + v * c11);$</p> <p>If $v < cThr$ then</p> <p style="padding-left: 20px;">$vacuum := s * \exp(c20 + v * (c21 + v * c22));$</p> | one 230 l/sec ion pump |
| 2 | <p>If $v \geq dThr$ then</p> <p style="padding-left: 20px;">$vacuum := s * \exp(d10 + v * d11);$</p> <p>If $v < dThr$ then</p> <p style="padding-left: 20px;">$vacuum := s * \exp(d20 + v * (d21 + v * d22));$</p> | two 230 l/sec ion pumps |
| 3 | <p style="text-align: center;">30 l/sec ion pump</p> <p style="padding-left: 20px;">$vacuum := s * \exp(t10 + v * t11) + \exp(t20 + v * t21);$</p> | |

Units of vacuum are 10^{-9} Torr, so $s=1.0E9$.

Coefficients used initially in the program are as follows:

```
cThr= 3.1; c10=-10.568; c11=-1.429; c20=10.972; c21=-14.797; c22=2.070;
dThr= 3.2; d10=-11.313; d11=-1.405; d20=-9.432; d21=-0.118; d22= 0.577;
t11= -1.535; t10= 5.09*t11; t21= -38.4; t20= -2.38*t21;
```

Of course, additional formula indexes can be defined using more constants by changing the local application source code.

In the initial implementation, readings are updated at 15 Hz. For 16 channels of conversion of 230 l/sec ion pumps, about 2.5 ms are required for the 68020-based local stations. Since these are vacuum readings, a more leisurely update rate could be adopted to save time. Also, since the vacuum signals from node 627 are also wired to the individual klystron stations in nodes 620–626, each station could run its own copy of VACT to spread the computing load.

The source code is 250 lines of Pascal, executing in 7K bytes.